UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO

JOÃO CARLOS PINHEIRO

PROCESSAMENTO DE CONSULTA EM UM *FRAMEWORK* BASEADO EM MEDIADOR PARA INTEGRAÇÃO DE DADOS NO PADRÃO DE *LINKED DATA*

FORTALEZA - CE
Setembro / 2011

JOÃO CARLOS PINHEIRO

PROCESSAMENTO DE CONSULTA EM UM *FRAMEWORK* BASEADO EM MEDIADOR PARA INTEGRAÇÃO DE DADOS NO PADRÃO DE *LINKED DATA*

Tese submetida à Coordenação do Curso Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará como requisito parcial para a obtenção do título de Doutor em Ciência da Computação.

Área de concentração: Ciência da Computação

Orientadora: Profª. Vânia Maria Ponte Vidal, Dra.
Co-orientador: Profº. José Antônio. F. de Macedo, Dr.

FORTALEZA - CE
Setembro / 2011

PROCESSAMENTO DE CONSULTA EM UM FRAMEWORK BASEADO EM MEDIADOR PARA INTEGRAÇÃO DE DADOS NO PADRÃO DE LINKED DATA

João Carlos Pinheiro

Tese de Doutorado apresentada ao Programa de Mestrado e Doutorado em Ciência da Computação da Universidade Federal do Ceará, como parte dos Requisitos para a obtenção do Grau de Doutor em Ciência da Computação do aluno João Carlos Pinheiro.

Composição da Banca Examinadora:

_____
Profa. Dra. Vânia Maria Ponte Vidal (Presidente) (DC/UFC)

_____
Prof. Dr. Marco Antonio Casanova (PUC/Rio)

_____
Prof. Dr. Fábio André Machado Porto (LNCC/Rio)

_____
Prof. Dr. Javam de Castro Machado (DC/UFC)

_____
Prof. Dr. José Antônio Fernandes de Macêdo (DC/UFC)

Aprovada em 26 de setembro de 2011

AGRADECIMENTOS

Pesquisar é como navegar num rio de curso desconhecido, inclui erros e riscos, não há como fazer previsões precisas, ou predeterminar um tempo para cumprir um objetivo. No entanto, ao fazer pesquisa, em algum momento o pesquisador deverá render-se à realidade e conviver com as limitações de tempo e recurso, reconhecendo limites que o próprio homem impõe ao desenvolvimento da ciência. Mesmo quando há mais experimentos a fazer, mais referências a buscar e mais casos a verificar, em algum momento é preciso escrever algo e se contentar com aquilo.

Isso também é verdade para o reconhecimento daqueles que ajudaram, direta ou indiretamente, na realização deste trabalho. Há sempre alguém esquecido e talvez alguma forma mais bela de agradecer, mas em algum momento temos de escrever, mesmo sabendo dos riscos imputados pelos limites de nossa memória e habilidade literária. Dentro dessas limitações, aqui vão meus agradecimentos.

A Deus, por tudo e principalmente por ter iluminado meu caminho durante essa oportunidade de crescimento e aprendizado.

A minha esposa, Tânia, que sempre me apóia e acredita em mim. E aos meus queridos filhos. Davi e Júlia, por tornar a vida tão interessante e alegre e saber respeitar este trabalho, mesmo sem compreender-lhe o significado. Sem esquecer quem deixou saudades, minha mãe, Maria do Socorro.

Ao meu padrasto, Ozino, e meus irmãos Jean, Daniel e Lia, pelo constante amor e encorajamento.

A minha orientadora Profª. Vânia Vidal, pela persistência, ensinamentos, dedicação à pesquisa e confiança demonstrada em meu trabalho.

Ao meu co-orientador, Profº. José Antônio. F. de Macedo, minha gratidão pelo incentivo e sábios conselhos.

Ao Profº. Fabio Porto, pela disponibilidade e opiniões seguras que ajudaram a aperfeiçoar este trabalho.

Ao Profº. Marco Antônio Casanova, por quem tenho grande admiração, cujas proficientes sugestões ajudaram a melhorar este trabalho. Apesar das poucas reuniões que tivemos, foi um privilégio para mim.

Aos amigos Marcel e Eveline, companheiros de importantes momentos durante esse período, pelas conversas, conselhos e aprendizado. Igualmente a Fernando Lemos, por seu vasto conhecimento e generosidade.

Aos demais amigos e colegas da UFC, Ângela, Bruno, Clayton, Danusa, Fernando Wagner, Hélio, Lígia, Lívia, Regis, Ticianne, Ticiana e todos que já passaram pela minha vida e me ajudaram com sua amizade, ideias e também conselhos.

À Universidade Federal do Ceará, pela oportunidade de grande crescimento profissional e intelectual.

Aos professores e funcionários do Departamento de Computação, em especial à Profª. Berna, pela simpatia e carisma, e ao Orley, da seção de Pós-Graduação do MDCC-UFC, pela eficiência com que sempre me atendeu, e, também, a Rosana, pelo alto-astral e descontração.

Ao Departamento Acadêmico de Informática do IFMA, pelo meu afastamento.

A meus colegas do Departamento Acadêmico de Informática do Instituto Federal do Maranhão, a quem devo sinceros agradecimentos pelo incentivo, André, Astor, Carla, Eva, Eveline, Gentil, Helder, Josenildo, Lourdinha (pedagoga), Luna, Márcio, Mauro, Nunes, Omar, Rafael, Salete e Santiago. Em especial à coordenadora, Profª. Karla Fook, pela compreensão na reta final de minha pesquisa, e a Jeane, pelas sugestões que ajudaram a aperfeiçoar este trabalho.

Agradeço também à FUNCAP e à CAPES, pelo apoio financeiro dado a este trabalho.

RESUMO

A Web evoluiu de um espaço de informação global de hipertexto para uma rede de *Linked Data* conhecida como a Web dos dados. Sendo que o uso de RDF, um dos pilares da Web semântica, tem sido fundamental para armazenamento e publicação de dados no padrão de *Linked Data* acessível via SPARQL *endpoint* através da linguagem de consulta SPARQL, que permite responder a consultas distribuídas que não poderiam ser respondidas por uma única fonte de dados ou até mesmo por motores de busca na Web. Porém, a dificuldade para formulação de consultas distribuídas tem sido um obstáculo para aproveitar esses dados em virtude da autonomia, distribuição e vocabulário heterogêneo das fontes de dados. Esse cenário ratifica a necessidade de mecanismos eficientes para a integração de dados, que podem potencializar o reuso desses dados de maneira simples e eficiente. Nesse contexto, este trabalho apresenta um *framework* baseado em mediador para integração de dados no padrão de *Linked Data* acessíveis via SPARQL *endpoint* em que o esquema global é representado por uma ontologia de domínio, que fornece um vocabulário compartilhado. Cada fonte de dados é descrita por uma ontologia de aplicação, que se refere ao mesmo vocabulário compartilhado da ontologia de domínio. Dentro desse escopo, este trabalho propõe um método para o processamento distribuído de consultas SPARQL, destacando-se: a) um algoritmo de reformulação de consulta em que duas questões-chave são tratadas: a busca de dados apenas em fontes de dados que podem contribuir com qualquer resultado intermediário, sem precisar recorrer a mecanismos de inferência para fazer a expansão da consulta; e a utilização de ligações *same-as* e *URI-links* para lidar com informação incompleta; b) na etapa de execução exploram-se algoritmos e técnicas que possibilitam a redução do volume de dados intermediários, o processamento paralelo de consultas, os modelos *pull* e *push* de entrega de dados e o processamento adaptativo que combina com proficiência os algoritmos junção. Essas técnicas são essenciais no ambiente altamente dinâmico dos *Linked Data*, que apresentam duas características que desafiam a avaliação de consultas SPARQL distribuídas: larga escala e imprevisibilidade nos tempos de entrega de dados. A estratégia de execução foi avaliado por meio de vários experimentos, e os resultados fornecem evidências empíricas de escalabilidade e ganho de desempenho para integração de dados.

Palavras-chave: Web Semântica. Ontologias. *Linked Data*. Integração de Dados. Mapeamentos. Processamento Distribuído de Consulta. RDF. SPARQL.

ABSTRACT

The Web evolved from a global information space of hypertext to the *Linked Data* network, also known as Web of Data. The use of RDF, one of the cornerstones of the Semantic Web, has been crucial for storage and publication of *Linked Data* accessible via SPARQL endpoint through the SPARQL query language, that allows answering distributed queries which could not be answered by a single data source or even search engines on the Web. However the difficulty of distributed query formulation has been an obstacle to take advantage of these data because of the autonomy, distribution and heterogeneous vocabulary of data sources. This scenario confirms the need for efficient mechanisms for data integration that can leverage the reuse of such data simply and efficiently. In that context, this work presents a *framework* based on a mediator for *Linked Data* integration accessible via SPARQL endpoint where global schema is represented by a domain ontology, which provides a shared vocabulary. Each data source, published on the Web according to the *Linked Data* principles, is described by an application ontology, whose vocabulary is restricted to be a subset of the domain ontology vocabulary. Inside this context, this work proposes a method for processing distributed SPARQL queries, including: a) an algorithm for query reformulation in which two key questions are addressed: the search for data only to data sources that may contribute with any intermediate result, without appeal to inference mechanisms for query expansion, and the use of same-as and URI-links to deal with incomplete information, b) the execution step explores algorithms and techniques that enable the reduction in the volume of intermediate data, parallel query processing, pull and push models for delivery of data and processing that combines adaptive join algorithms proficiently. These techniques are essential in the highly dynamic environment of the *Linked Data*, which have two characteristics that challenge the distributed SPARQL query evaluation: a large scale and unpredictability in time data delivery. The optimization strategy was evaluated through several experiments, and the results provide empirical evidence of its scalability and performance gains for data integration.

Keywords: Semantic Web. Ontologies. Linked Data. Data Integration. Schema Mappings. Distributed Query Processing. RDF. SPARQL.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

API	Application Programming Interface
CSV	Comma-Separated Values
GAV	Global-As-View
DARQ	Distributed ARQ
HTTP	Hypertext Transfer Protocol
JDBC	Java Database Connectivity
LAV	Local-As-View
OWL	Web Ontology Language
PHJ	Pipelining Hash Join
QEEF	Query Engine Execution Framework
RDF	Resource Description Framework
RDFS	RDF Schema
SBJ	Set-Bind-Join
SOAP	Simple Object Access Protocol
SGBD	Sistema de Gerenciamento de Banco de Dados
SemWIQ	Semantic Web Integrator and Query Engine
SPARQL	SPARQL Protocol and RDF Query Language
UML	Unified Modeling Language
URI	Uniform Resource Identifier
XML	Extensible Markup Language
WWW	World Wide Web
W3C	World Wide Web Consortium

1 INTRODUÇÃO

Neste capítulo, introduz-se o contexto onde está inserida esta tese e se sintetizam suas contribuições. Para tal, apresentam-se a motivação, o problema de pesquisa, os objetivos a ser alcançados, bem como a organização dos demais capítulos do documento.

1.1 Motivação

O surgimento recente de dados em padrões abertos, compreensíveis logicamente por máquinas e suportados pelo *World Wide Web Consortium* (W3C), tem impulsionado a publicação de dados na Web. Por outro lado, o consumo e reutilização desses dados ainda é algo difícil de ser feito devido ao vocabulário heterogêneo dos dados e a sua fragmentação e distribuição natural na Web. Este cenário traz consigo a necessidade de mecanismos eficientes à integração de dados, que podem potencializar o reuso desses dados de maneira eficaz.

No padrão *Linked Data,* especificamente, cada fragmento de dado no formato de triplas RDF (*Resource Description Framework*) descreve a si mesmo e suas relações, com outros fragmentos de dados de forma descentralizada. A expressão *Linked Data* se refere ao conjunto das melhores práticas de publicação e conexão de dados estruturados na Web que vêm sendo adotado cada vez mais por provedores de dados, levando à criação de um espaço global de dados que contém bilhões de triplas – a Web dos dados (BIZER; HEATH; BERNERS-LEE, 2009).

Esse padrão destaca-se como o principal representante das fontes modernas de dados. Hoje, empresas, governos e sociedade publicam dados nesse padrão, a fim de permitir a reutilização em ambientes heterogêneos e distribuídos por qualquer interessado no desenvolvimento de aplicações que utilizem tais dados. Um bom exemplo dessa iniciativa é a grande quantidade de dados governamentais disponíveis na Web, gerados a partir das iniciativas de *open-government*, que defendem a ampla divulgação de dados a cidadãos e organizações, facilitando a publicação e a reutilização de informações públicas como, por exemplo, informações financeira, econômica, dados populacionais e demográficos. Exemplos de sucesso desta iniciativa são os governos dos Estados Unidos e Inglaterra que disponibilizam serviços de acesso a dados governamentais (*http://services.data.gov/sparql*; *http://services.data.gov.uk/sparql*) publicados no padrão de *Linked Data.*

Em geral, os *Linked Data* são (BERNERS-LEE et al., 2006): abertos e não-proprietários – podem ser acessados por meio de uma variedade ilimitada de aplicações; modulares – podem ser combinados com quaisquer outros *Linked Data* sem planejamento prévio; escaláveis – é fácil adicionar mais *Linked Data* aos já existentes, mesmo quando os termos e definições utilizadas mudam ao longo do tempo. Conforme Berners-Lee (2006), em essência, os *Linked Data* são inspirados no sucesso da Web *de documentos,* envolvendo padronização da semântica por trás dos dados.

Aplicações podem acessar *Linked Data* na Web através de consultas a um SPARQL *endpoint*. Embora esse acesso possa prover dados valiosos para a aplicação, essa abordagem ignora o grande potencial dos *Linked Data*, pois não explora as possibilidades deste espaço de dados global que integra um grande número de fontes de dados interligados (MAGALHÃES et al., 2011). Com advento do SPARQL 1.1, é possível formular uma consulta federada que busque informações em múltiplas fontes de dados, desde que se conheça previamente a localização, o vocabulário heterogêneo e como combinar os dados dessas fontes. Hartig e Langegger (2010) classificam dois tipos duas abordagens para realizar essas consultas sobre os Linked Data: tradicionais e inovadoras.

Em relação às abordagens *tradicionais*, tem-se *data warehousing* e federação de consultas que são abordagens amplamente discutidas na literatura de banco de dados para realização de consultas sobre dados distribuídos em fontes autônomas. Consultas sobre os *Linked Data* podem utilizar essas abordagens tradicionais que requerem o conhecimento prévio das fontes de dados relevantes. A seguir descrevem-se a aplicação dessas abordagens.

No contexto de *Linked Data*, pode-se materializar dados das fontes relevantes em um *data warehouse*. Assim, quando uma consulta é formulada, ela é executada sobre os dados que se encontram no repositório, não havendo recuperação direta das fontes de dados. Isso pode implicar em inconsistência entre o repositório e as fontes de dados que são autônomas e evoluem rapidamente, podendo gerar dados desatualizados. Em fontes cujo volume de dados é muito grande, como é o caso do *DBpedia* (*http://dbpedia.org/*), a materialização dos dados tende a requerer bastante tempo e espaço de armazenamento. Por outro lado, se ganha velocidade na obtenção das respostas, pois não há necessidade de comunicação na rede.

A *federação de consultas* baseia-se na distribuição do processamento de consultas para múltiplas fontes de dados autônomas. O objetivo é dar ao usuário acesso aos dados por meio de algum vocabulário padrão especificado em uma ontologia de domínio. Consultas podem ser formuladas baseadas na ontologia de domínio e um mediador transparentemente

decompõe a consulta em subconsultas, encaminha as subconsultas a múltiplos serviços de consulta distribuídos e, após os resultados obtidos, os dados são integrados e a resposta final é devolvida ao usuário. Uma vantagem dessa abordagem é que ela não requer tempo ou espaço adicional para materialização de dados e obtêm-se os dados atualizados, pois os mesmos são extraídos das fontes no momento em que são requisitados pelas consultas, apesar de ineficiente frente a transmissões de rede necessárias para realização das subconsultas sobre as fontes de dados. DARQ (QUILITZ; LESER, 2008) é um mediador baseado no processador de consultas Jena ARQ capaz de realizar consultas distribuídas sobre a web dados. SemWIQ (LANGEGGER, 2010) é outro mediador que estende o Jena ARQ a fim de consultar a web de dados fazendo uso de estatísticas (LANGEGGER; WOSS, 2009) para otimizar as consultas. Desafios relacionados à eficiência de consultas federadas e uma abordagem para otimização dessas consultas baseada em programação dinâmica foram tratados por (OLAF e STAAB, 2011).

No enfoque exploratório proposto por Hartig et al. (2009), as fontes de dados que podem contribuir para a resposta do usuário são descobertas e recuperados em tempo de execução da consulta. Consultas exploratórias seguem RDF *links* para obter mais informações sobre os dados já existentes. Através do uso de dados recuperados a partir das URIs usadas em uma consulta como ponto de partida, o processador de consultas avalia partes da consulta. Este enfoque possui duas limitações: recupera apenas URIs e exige que a consulta seja executada a partir de uma URI somente.

Neste trabalho, a abordagem de federação de consultas baseada em mediadores é mais adequada, considerando que os dados estão disponíveis em escala global na Web – no padrão de *Linked Data* – podendo ser acessados e atualizados livremente. A seguir apresentam-se os componentes básicos e funcionalidades utilizadas neste trabalho que possibilitam a integração de dados publicados no padrão de *Linked Data*:

a) *esquema de mediação* (*global*) – pesquisas recentes têm utilizado ontologias para especificar o esquema de mediação no âmbito da integração de dados (AMANN et al., 2002; CRUZ; XIAO; HSU, 2004; CALVANESE et al., 2008; LANGEGGER, 2010; PINHEIRO et al., 2010). Neste trabalho, utiliza-se uma ontologia denominada ontologia de domínio para definir um vocabulário padrão e separar o conhecimento do domínio do conhecimento operacional;
b) *fontes de dados* – devem aderir ao padrão de *Linked Data,* que usa tecnologias da Web semântica para publicar dados estruturados na Web e criar ligações entre

dados de diferentes fontes. Os *Linked Data* têm por base alguns padrões bem estabelecidos, por exemplo: o mesmo mecanismo de identificação usado para localizar recursos na Web (URIs) e um modelo de dados comum (RDF). Os dados publicados devem estar nesse modelo ou serem convertidos ao mesmo;

c) *linguagem de consulta declarativa* – é necessária para a formulação consistente de consultas complexas, neste trabalho se utiliza a linguagem SPARQL, que atualmente é a linguagem franca para consultar *Linked Data*. O W3C (W3C, 2002), órgão responsável pela padronização do uso do SPARQL, está elaborando uma extensão associada à federação de consultas SPARQL, e o *draft* foi publicada pelo W3C recentemente (PRUD'HOMMEAUX; BUIL-ARANDA, 2011);

d) *mapeamentos* – são necessários entre os esquemas (vocabulários) heterogêneos das fontes de dados no padrão de *Linked Data* e o esquema de mediação para reformulação correta das consultas;

e) *reformulação da consulta* – as consultas formuladas sobre o esquema de mediação são reescritas em subconsultas de acordo com os mapeamentos, e então executadas nas fontes de dados. As subconsultas constituem o plano de execução originalmente submetido ao esquema de domínio. Neste trabalho, durante a reformulação da consulta, há a preocupação de melhorar a qualidade e a quantidade dos resultados obtidos;

f) *execução de consulta distribuída* – o contexto sobre o qual o sistema de integração proposto neste trabalho opera é altamente dinâmico, um otimizador de consultas tem muito menos informação disponível do que num ambiente tradicional de banco de dados. Como resultado, duas situações podem acontecer: o otimizador pode não ter informação suficiente para decidir sobre um bom plano de execução ou um plano que, a princípio, pareça bom, pode não o ser se as fontes não responderem exatamente da forma esperada. Arenas e Perez (2011) destacam que o processamento de consultas SPARQL distribuídas sobre *Linked Data* são tarefas bastante complexas. Halevy, Rajaraman e Ordille (2006) realizaram retrospectiva sobre dez anos de pesquisa na área de integração de dados que apontam o processamento de consultas adaptativo, dentre os temas de pesquisa fundamentais para a solução de problemas de integração. Muitos estudos (AVNUR; HELLERSTEIN, 2000; HELLERSTEIN et al., 2000; IVES;

HALEVY; WELD, 2004; AYRES; PORTO; MELO, 2003) têm sido realizados com o intuito de tornar a execução da consulta o mais adaptativa possível.

g) *tradutores* (*wrappers*) – os SPARQL *endpoints* são serviços de consulta distribuídos que utilizam o protocolo HTTP e podem ser apontados como candidatos naturais para acesso aos dados no padrão de *Linked Data* na web. Neste trabalho, não há preocupação em configurar *wrappers* específicos ou fazer tradução de dados, pois o SPARQL já define um protocolo desse tipo (CLARK; FEIGENBAUM; TORRES, 2008), incluindo formatos de resultado (BECKETT; BROEKSTRA, 2006).

1.2 Formulação do problema

A área de pesquisa deste trabalho é a integração de dados no padrão de *Linked Data* para aplicações de domínio específico. A título de ilustração, apresenta-se um exemplo de um cenário típico do problema de integração a partir de múltiplas fontes de dados públicas no padrão de *Linked Data* de domínio médico (BIZER et al. 2011) que incluem informações sobre doenças (*Diseasome*), drogas (*DrugBank*), bulas de drogas (*DailyMed*), informações sobre medicamentos e efeitos adversos (*Sider*) e, por fim, uma fonte de dados com temas variados (*DBpedia*). A Figura 1 apresenta um esquema minimalista das fontes de dados com as respectivas ligações.

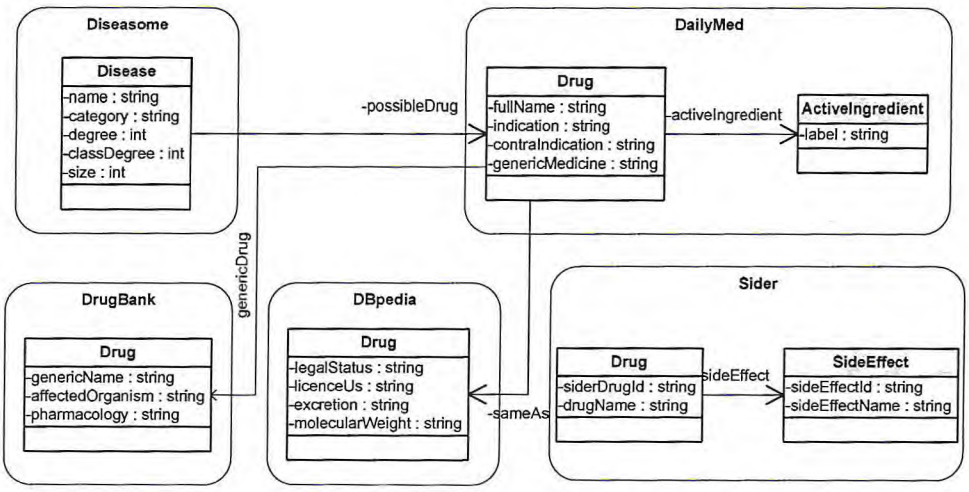


Figura 1 – Fontes de dados do domínio médico no padrão de *Linked Data*

Se o usuário tiver conhecimento prévio do vocabulário das fontes de dados e da sintaxe do SPARQL para consultas federadas, é possível formular uma consulta informando explicitamente os SPARQL *endpoints*.

Como exemplo de consulta, considere-se a necessidade de saber as possíveis drogas para a doença de *Alzheimer*. Com base no esquema apresentado na Figura 1, é necessário formular uma consulta federada que busque informações nas fontes de dados *Diseasome* e *DailyMed*, ilustradas na Figura 2.

```
PREFIX diseasome:<http://www4.wiwiss.fu-berlin.de/diseasome/.../>
PREFIX dailymed: <http://www4.wiwiss.fu-berlin.de/.../dailymed/>
SELECT DISTINCT ?dgn
WHERE {
 SERVICE <http://www4.wiwiss.fu-berlin.de/diseasome/sparql> {
 ?d diseasome:name ?n
 FILTER regex(?n, 'Alzheimer')
 ?d diseasome:possibleDrug ?dg .
 }
 SERVICE <http://www4.wiwiss.fu-berlin.de/dailymed/sparql> {
 ?dg dailymed:fullName ?dgn .
 }}
```

Figura 2 – Consultas federadas sobre as fontes de dados *Deseasome* e *DailyMed*

Esta abordagem traz alguns inconvenientes para o usuário, pois para conseguir definir consultas especificando *o que* se deseja saber e *onde* os dados podem ser encontrados o usuário deve, previamente, ter conhecimento abrangente de consultas federadas e, também, conhecer a localização e o vocabulário heterogêneo das fontes de dados.

A dificuldade para formulação de consultas distribuídas pode aumentar se o número de fontes de dados a ser consultadas for ainda mais abrangente. Por exemplo, considere que o usuário deseja, além das possíveis drogas para a doença de Alzheimer, também obter informações sobre indicação e ingredientes ativos (*DailyMed*), o nome genérico da droga (*DrugBank*) e informações sobre a legalidade (*DBPedia*). A consulta fica bem mais desafiadora, pois o usuário precisa buscar informações adicionais nas fontes de dados *DrugBank* e *DBPedia* e definir com precisão os predicados de junção. Além do mais, o usuário precisa lidar com vocabulário heterogêneo das fontes de dados. Um exemplo é o caso das propriedades *fullName* e *drugName*, que apesar de possuírem nomes diferentes correspondem a informação equivalentes, enquanto *genericName* corresponde a outra informação, que é o nome genérico do medicamento.

Esse cenário corrobora a necessidade de mecanismos eficientes à integração de dados, que pode potencializar o reuso dos dados de maneira simples e eficiente de forma a superar os problemas de heterogeneidade, autonomia e localização de fontes de dados.

Um problema não tratado pelas propostas atuais com objetivos similares ao deste estudo, até onde se conhece, é que na ausência de *URI-links* e propriedades *sameAs* entre as fontes de dados, o algoritmo de reformulação não consegue identificar operações de junção, consequentemente não retornam informações completas. Conforme ilustrado na Figura 1, as fontes de dados *DailyMed* e *Sider* refletem o problema, em que as propriedades *fullName* e *drugName* corresponderem ao nome das drogas, que, apesar de não possuir ligação, são comuns em ambas as fontes de dados.

Outro problema a ser considerado é o processamento de consultas distribuídas sobre diversas fontes de dados publicadas e atualizadas de acordo com os princípios dos *Linked Data*, que tem gerado um volume crescente de dados e, consequentemente, uma demanda por seu consumo. Ademais, é preciso considerar a ausência de estatísticas e lidar com a autonomia e imprevisibilidade dos SPARQL *endpoints* acessados livremente na Web, que podem, às vezes, ser lentos para responder caso a demanda de vários clientes seja elevada ou a manutenção/atualização de dados esteja ocorrendo na fonte de dados. Assim, podem chegar com diferentes tempos de resposta, além do volume de dados recuperados serem, geralmente, desconhecidos.

A execução de consultas SPARQL distribuída ainda está em estágio inicial e indica a necessidade de uma base tecnológica escalável (OLAF e STAAB, 2011) que possa capturar eventos imprevisíveis durante a execução das subconsultas. Isso sugere uma estratégia adaptativa e que considere algoritmos adequados para lidar com atrasos na chegada de dados ao mediador, bem como a geração de resultados o mais cedo possível sem negligenciar a sobrecarga de comunicação, o principal custo nos sistemas baseados em mediadores.

1.3 Resumo da proposta e objetivos

O presente trabalho tem como objetivo geral contribuir com uma solução mais simples e eficiente para a integração de dados, conforme a problemática apresentada na Seção 1.2.

Definiu-se um *framework*, baseado em trabalhos anteriores (VIDAL et al., 2009; PINHEIRO, et al., 2009; PINHEIRO, et al., 2010), para integração de dados no padrão de *Linked Data* baseados em mediadores e que utiliza ontologia como esquema de mediação. O *framework* adota uma arquitetura de três níveis de esquemas, conforme exibido na Figura 3,

em que para cada fonte de dados publicada no padrão de *Linked Data* existe uma ontologia de aplicação, que corresponde a um fragmento homogêneo da ontologia de domínio e ajuda a dividir o problema de processamento de consultas em dois subproblemas (menos desafiadores), conforme segue:

a) realizar uma decomposição (*unfolding*) simples de consultas sobre a ontologia de domínio em subconsultas sobre as ontologias de aplicação, isto é, como reformular uma consulta sobre a ontologia de domínio em uma ou mais subconsultas expressas em termos das ontologias de aplicação sem a necessidade de encontrar mapeamentos em tempo de execução;

b) como responder a consultas no contexto de *data-exchange* (LIBKIN; SIRANGELO, 2011), ou seja, como transformar uma subconsulta sobre uma ontologia de aplicação em uma subconsulta sobre uma fonte de dados no padrão de *Linked Data*.

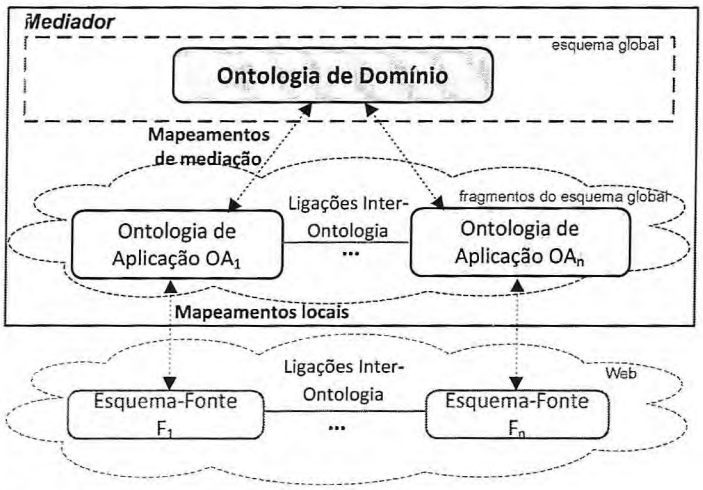


Figura 3 – arquitetura de três níveis de esquemas

A arquitetura de três níveis se mostra adequada aos nossos objetivos, uma vez que melhor reflete a realidade de contarmos com fontes de dados heterogêneas e simplificar a integração de dados. Para lidar com informação incompleta, ligações interontologia são consideradas durante a reformulação da consulta, o que gera resultados mais significativos e completos para o usuário.

Outro problema abordado neste trabalho é o de execução de consultas SPARQL distribuídas. Para cumprir esse objetivo, foram analisadas técnicas conhecidas envolvendo operações de junção que exploram a natureza dos *Linked Data*, destacando-se a redução do volume de dados intermediários e o processamento paralelo de consulta. Apresentou-se uma estratégia adaptativa que possibilita mudar, em tempo de execução, o algoritmo de junção recorrendo aos algoritmos mais adequados a cada instante.

Para o processamento de consultas SPARQL propõe-se um método para processamento de consultas cujos passos podem ser, assim, resumidos: a) a consulta sobre o esquema de mediação é transformada numa árvore que representa a estrutura da consulta; b) a árvore gerada é reformulada em uma combinação de subconsultas sobre as fontes de dados relevantes com eliminação das fontes de dados não-relevantes para o resultado da consulta; e, por fim, c) tem-se a etapa de execução de subconsultas e a composição do resultado final.

É importante observar que a proposta deste trabalho está focalizada no problema de integração de fontes de dados no padrão de *Linked Data*, que são previamente conhecidas por um projetista. Não há descoberta automática de fontes de dados. A proposta é aplicável, por exemplo, quando se deseja integrar dados publicados no padrão de *Linked Data* de entidades governamentais distribuídos por vários municípios ou Estados; ou de uma instituição com dados operacionais distribuídos em filiais. O projetista é responsável por especificar a ontologia de domínio. A geração das ontologias de aplicação e a geração dos mapeamentos são tratados em outro trabalho do grupo (SACRAMENTO, et al., 2010).. Está fora do escopo deste trabalho uma etapa de otimização das subconsultas SPARQL enviadas para execução nos SPARQL *endpoints* (otimização local), ficando, a critério dos sistemas que armazenam os *Linked Data*, a realização desta otimização.

1.4 Contribuições

Esta tese pretende contribuir para um melhor entendimento dos aspectos relacionados à integração de dados no padrão de *Linked Data*. Em síntese, a reformulação de consulta e a estratégia de execução constituem as contribuições-chave deste trabalho, que visa ajudar a preencher uma importante lacuna, num campo ainda de certa forma pouco explorado.

A seguir, pontuam-se as principais contribuições.

a) Desenvolvimento de um algoritmo para reformulação de consulta que utiliza

ligações *interontologias*, um importante diferencial deste trabalho, pois, como enunciado anteriormente, melhora a qualidade e quantidade dos resultados obtidos.

b) Algoritmos de junção que exploram paralelismo e a redução na transmissão de dados intermediários ao mediador:
   i. definiu-se o algoritmo *set-bind-join*, cujo principal objetivo é reduzir a transmissão de dados intermediários para o mediador. Este algoritmo explora o paralelismo intraoperador visando melhorar a eficiência no processamento das consultas;
   ii. revisitou-se o algoritmo *pipelining hash-join* proposta por Wilschut e Apers (1993) para banco de dados paralelos. Este algoritmo possui duas características relevantes no acesso aos dados no padrão de *Linked Data*: 1) produzir resultados incrementalmente à medida que ficam disponíveis; 2) permitir execução contínua do algoritmo mesmo quando ambos *SPARQL endpoints* estiverem em atraso.

c) Definição de uma estratégia adaptativa para o processamento de consulta para mudar o algoritmo de junção em tempo de execução.

d) Avaliação do impacto dos algoritmos de junção no desempenho geral do processamento de consulta.

e) Definição de um método para o processamento distribuído de consulta SPARQL no *framework* proposto.

f) Revisão bibliográfica que mostra o estado da arte em integração de dados no padrão de *Linked Data*.

1.5 Organização dos capítulos

Os estudos e análises realizados no desenvolvimento desta pesquisa estão estruturados em quatro partes organizados em capítulos ilustrados na Figura 4 a seguir:

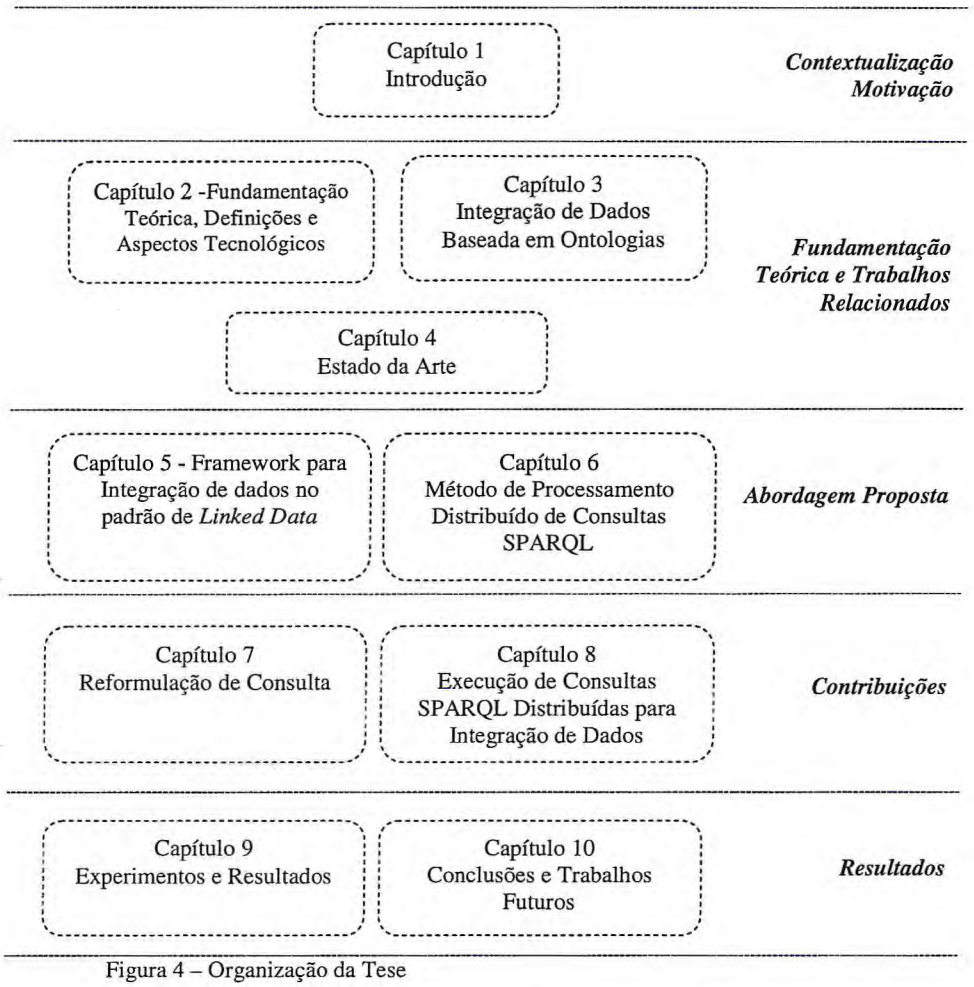


Figura 4 – Organização da Tese

No Capítulo 2, faz-se revisão dos conceitos e definições essenciais ao entendimento do tema e apresentam-se as principais tecnologias e padrões relacionados à Web semântica, que permitem a disseminação dos *Linked Data.*

O Capítulo 3 se destina a prover uma revisão teórica relacionada à integração de dados relevantes neste trabalho, em que se destacam a arquitetura de mediadores, o uso de ontologias para integração de dados e os enfoques para definição de mapeamentos entre ontologias. O objetivo é fornecer ao leitor fundamentos teóricos antes da apresentação do *framework* e do método proposto para o processamento distribuído de consultas SPARQL.

No Capítulo 4 é feita revisão dos principais trabalhos na área. Faz-se análise comparativa considerando as etapas comuns no processamento de consultas em sistemas de integração de dados: reformulação de consulta, execução e consolidação dos resultados finais.

O Capítulo 5 apresenta a arquitetura do *framework* proposto para integração de dados no padrão de *Linked Data* concomitantemente com um exemplo e discute-se como

ajudar a resolver algumas das limitações das atuais abordagens de integração de dados baseada em ontologia.

No Capítulo 6 são apresentadas as etapas do método de processamento distribuído de consultas ilustradas por meio de um exemplo, compreendendo as etapas de tradução da consulta; reformulação; execução e composição do resultado final.

O Capítulo 7 apresenta o algoritmo de reformulação de consulta que utiliza propriedade e ligações *interontologias* durante a reformulação da consulta para obter resultados mais completos juntamente com as regras de correção para validar o algoritmo proposto.

O Capítulo 8 trata de execução de consultas distribuídas e técnicas conhecidas envolvendo operações de junção que explora a natureza dos *Linked Data*, destacando-se redução do volume de dados intermediários, processamento paralelo do plano de execução, modelos *pull* e *push* para entrega de dados, um modelo de notificação de eventos baseado no padrão de projeto *Observer*. E define-se uma estratégia que possibilita mudar em tempo de execução o algoritmo de junção.

O Capítulo 9 fornece experiências da abordagem proposta e os resultados obtidos. Descreve a metodologia de testes utilizada, fontes de dados, e consultas utilizadas nos experimentos. Os resultados são discutidos e apresentados graficamente.

Finalmente, o Capítulo 10 resume o trabalho proposto, discute as contribuições alcançadas e indica algumas direções em que a pesquisa apresentada pode ser estendida.

2 FUNDAMENTAÇÃO TEÓRICA, DEFINIÇÕES E ASPECTOS TECNOLÓGICOS

Neste capítulo, serão vistos conceitos, definições essenciais ao entendimento do trabalho e principais tecnologias e padrões da Web semântica, cujos conceitos constituem a base teórica que permite a disseminação dos *Linked Data*. Primeiro, apresentam-se as ontologias para a resolução de conflitos de interoperabilidade semântica no contexto integração de dados; em seguida, analisam-se os padrões da Web semântica, modelo de dados RDF, esquema RDF e linguagem OWL. Os padrões possibilitam a publicação dos dados no padrão de *Linked Data,* que é apresentado na seção 2.7. E, na seção 2.8, estão as tecnologias que possibilitam consultas a esses dados, como SPARQL (linguagem de consulta, protocolo de acesso e formatação de resultados) juntamente com os SPARQL *endpoints*. Por fim, destaca-se uma recente extensão da linguagem SPARQL para execução de consultas federadas, que se tem apresentado como grande diferencial no redirecionamento de pesquisas relacionadas à área de integração de dados no padrão de *Linked Data*.

2.1 Introdução

O advento da publicação de dados na Web no padrão de *Linked Data* proporciona às pessoas e organizações inúmeras facilidades para o compartilhamento de dados, incentivado fortemente pela publicação de dados na Web e pela *triplificação* (*triplify*[1]) de dados provenientes de fonte de dados relacionais (AUER et al, 2009). Entretanto, esse padrão impôs novos desafios à pesquisa sobre integração de dados.

Os *Linked Data* já são numerosos e altamente heterogêneos (BIZER; HEATH; BERNERS-LEE, 2009), cujas diversas interfaces de acesso apresentam capacidades de processamento bastante distintas. Os SPARQL *endpoints* têm sido fundamentais a sua proliferação, pois permitem ao usuário (humano ou interface de programação) consultar RDF/OWL por meio da linguagem SPARQL, que é a recomendação do W3C mais promissora para consultar dados no padrão de *Linked Data*. Associados a essa linguagem, existem inúmeros padrões que tratam de aspectos como protocolos para formatação de resultados e um protocolo de comunicação. Esses padrões viabilizaram o surgimento de uma arquitetura para a publicação e consumo dos *Linked Data* de forma ubíqua.

[1] http://triplify.org/Overview

As áreas de pesquisas relevantes no contexto desta tese podem ser visualizadas na Figura 5. As ontologias e mapeamentos fornecem alternativa aos sistemas de integração de dados tradicionais. No lado direito da figura, os campos relacionados aos *Linked Data* são descritos. Eles incluem a linguagem SPARQL, RDF/OWL, SPARQL *endpoints,* bem como o processamento de consultas SPARQL federadas. No lado esquerdo da Figura 4 estão os principais tópicos relacionados à integração de dados, destacando-se sistemas distribuídos, arquitetura mediador-*wrapper*, processamento distribuído de consulta, mapeamentos e ontologias, que serão apresentados no Capítulo 3.

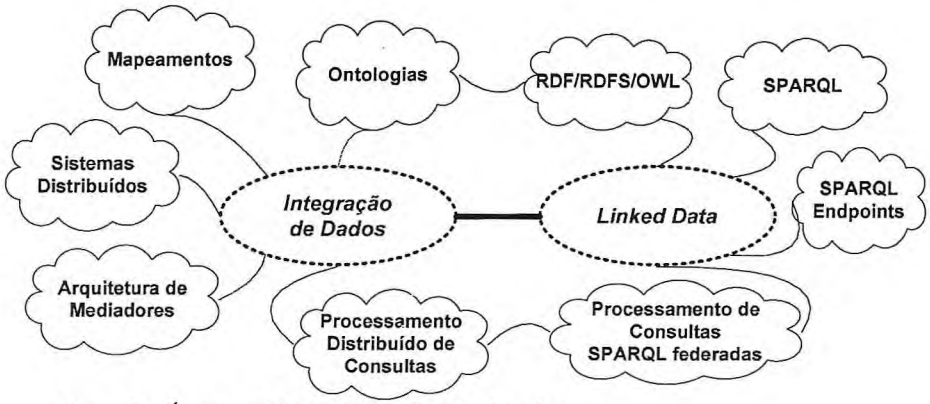


Figura 5 – Áreas e subáreas de pesquisa relacionadas

A seguir apresentam-se as tecnologias e conceitos relevantes no contexto deste trabalho.

2.2 Ontologias

Ontologias podem ser utilizadas por aplicações que necessitem compartilhar informações de determinado domínio de estudo. Para isso, utiliza-se um vocabulário específico e um conjunto de axiomas lógicos que não apenas fornecem a semântica pretendida aos termos desse vocabulário, mas também restringem a interpretação e utilização dos termos.

Uma definição clássica de ontologia é a proposta por Gruber (1993, p.199): "Ontologia é uma especificação formal de uma conceitualização". Essa definição foi ampliada por Fensel (2001, p.4): "Ontologia é uma especificação explícita de uma conceitualização e uma descrição formal dos conceitos e relacionamentos compartilhados de uma área de conhecimento". Conceitualização refere-se então a um modelo abstrato de um domínio:

explícita é concernente a definições de nomenclaturas não-ambíguas; **formal** significa passível de ser processada automaticamente; e **compartilhada** representa o conhecimento consensual de um domínio.

Em suma, pode-se inferir que ontologia é a representação de uma compreensão compartilhada de conceitos em um domínio específico de interesse, tal como acordado por uma comunidade. Essa representação deve ser clara, concisa e coerente para essa comunidade (NECIB, 2007), sendo que uma comunidade pode ser um grupo de pessoas ou sistemas de informação que interagem uns com os outros dentro de um domínio de interesse comum. Para construir uma ontologia é necessário lidar com um vocabulário de termos básicos, com uma especificação precisa do que tais termos significam e como se relacionam entre si.

Embora existam diferentes definições do termo *ontologia*, algumas noções básicas quanto à estrutura são compartilhadas pela maioria das abordagens. Considerando-se que uma ontologia pode ser representada por uma hierarquia de conceitos e de relações, a seguir são focalizados alguns elementos usados para representar as ontologias neste trabalho e seu significado (NOY; McGUINNESS, 2001):

a) Classe (também conhecida como *conceito*) – descrevem conceitos em um domínio. Por exemplo, uma classe de Vinho representa todos os vinhos. Uma classe pode ter subclasses que representam conceitos que são mais específicos do que a superclasse (NOY; McGUINNESS, 2001). Por exemplo, podemos dividir a classe de todos os vinhos em tinto, branco e rosé. Alternativamente, podemos dividir uma classe de todos os vinhos em espumantes e não-espumante.

b) Propriedade (também chamado de **predicado**) – vista como relação, uma vez que é usada para estabelecer um relacionamento entre dois termos. O primeiro termo deve ser um conceito que represente o domínio da relação; e o segundo termo deve ser um conceito que represente o contradomínio (*range*) da relação. Por exemplo, um *revisor* poderia ser representado como uma relação de tal forma que seu domínio é *pessoa* e o contradomínio é *revisões*. O contradomínio de uma propriedade também pode ser um tipo de dado primitivo como *string*, *decimal* ou *boolean*. E uma relação pode ter sub-relações.

c) Instância (indivíduo) – unidade materializada de uma classe, como *Maria* é uma

instância de *pessoa*, ou um *carro* específico que possui uma *placa* identificando-o unicamente.

A motivação principal do paradigma declarativo das ontologias é a modelagem de sistemas em alto nível e mais próximo do conhecimento do domínio a ser modelado. Consiste em uma forma mais flexível de descrever um domínio sem nenhum compromisso com sua implementação (FENSEL, 2001), destacando-se que há a garantia de que, nas ontologias, os dados são estruturados com vocabulário livre de ambiguidades e formalismo passível de processamento automático (NOY; McGUINNESS, 2001), o que torna o uso, em sistemas de integração de dados, muito apropriado.

A representação do conhecimento de ontologias deve ser realizada por linguagens específicas, normalmente variantes da lógica descritiva que provêem alta expressividade e raciocínio. Alguns exemplos, utilizados no contexto deste trabalho, são RDF, RDFS e OWL, que serão apresentados a seguir.

2.3 Resource Description Framework

O *Resource Description Framework* (RDF) de acordo com Brickley e Guha (2004) é um dos pilares para a construção de ontologias como mecanismo de compartilhamento e interpretação de dados em domínios diferentes. E pode ser interpretado em três níveis distintos de abstração (KLYNE; CARROLL, 2004): a) *nível sintático* – trata-se essencialmente de documento XML; b) *nível estrutural* – conjunto de triplas na forma (*sujeito-predicado-objeto*) que codificam fatos conhecidos. Os predicados codificam relacionamentos binários entre um sujeito e um objeto e estão rotulados com *Uniform Resource Identifier* (URI). Um sujeito é um recurso identificado por um espaço de nomes globais fornecido pelo uso de URI. Um *objeto* pode ser outro recurso relacionado, ou o valor da propriedade do sujeito; c) *nível semântico* – grafos dirigidos com semântica predefinida associada aos nós e arcos. Comumente, denomina-se *modelo de dados RDF,* devido à possibilidade de se construir ontologias de nível elementar, ou seja, representar relações entre recursos.

O ponto forte do modelo de dados RDF está na generalidade: estruturas de dados para publicação de dados no padrão de *Linked Data* facilmente mapeados para grafos rotulados dirigidos, característica central desse modelo de dados.

Sob a representação de grafo, um sujeito (recurso) é representado por uma *elipse* identificada por um URI, o objeto (valor) por *retângulo* e o predicado (propriedade) por um *arco* que conecta o sujeito ao objeto (vide Figura 6).

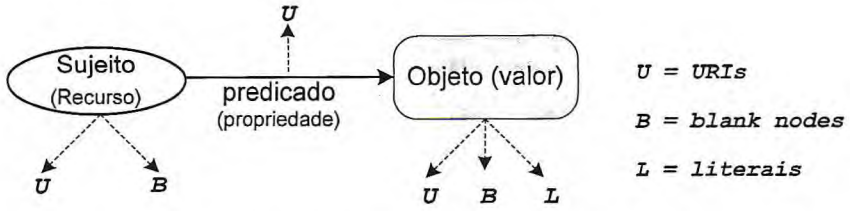


Figura 6 – Representação gráfica de uma tripla RDF

O uso de nomes globais é extremamente importante porque as triplas podem sempre ser combinadas (*merged*) sem tradução de nome. Grafos inteiros podem ser transportados e combinados sem tradução, fato que constitui grande vantagem para o intercâmbio e compartilhamento de dados, propiciando integração de dados. Sites como o *eBay* e *Amazon*, por exemplo, fazem uso do RDF na estrutura de pesquisa.

Conforme a Figura 6, *U, B e L* denotam conjuntos disjuntos de URIs , *blank nodes* e literais, respectivamente; e *UB, U e UBL* denotam $U \sqcup B$, *U, e* $U \sqcup B \sqcup L$ respectivamente. Na sequência, formalizam-se as noções de tripla RDF e grafos RDF.

Definição 1 (tripla RDF) – uma tripla RDF *t* é um conjunto de tuplas (*s, p, o*), em que *s*, *p* e *o* são sujeito, predicado e objeto respectivamente. Sendo que $s \in (U \sqcup B)$, $p \in U$ e $o \in (U \sqcup B \sqcup L)$. □

A Figura 7 traz a representação de uma tripla RDF. O prefixo *am* é utilizado como sinônimo para o espaço de nomes, identificado no URI *http://amazon.com/schema/*, no qual o predicado *data-revisão* fora definido.

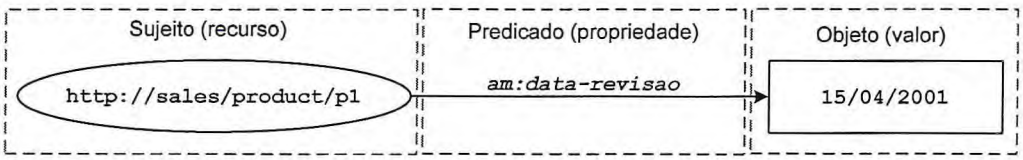


Figura 7 – Representação gráfica de uma tripla RDF

Definição 2 (grafo RDF) – um grafo RDF *G*, também conhecido por *Dataset*, é um conjunto de triplas RDF representado por um grafo direcionado, em que cada *tripla RDF* define e codifica relacionamentos binários entre um sujeito e um objeto rotulado por um predicado. □

A Figura 8 ilustra um grafo RDF que é um conjunto de asserções, por exemplo, o produto *P1* de título *Apple iPad*, foi revisado (propriedade de objeto *s:hasReviewer*) pelo revisor *foaf:joao* com comentário "*Wonderful Tablet*" e nota 8.5. Vê-se que nos nomes qualificados a seguir como prefixo "*rdf*", "*foaf*" e "*s*" correspondem, respectivamente, aos espaços de nomes *http://www.w3.org/1999/02/22-rdf-syntax-ns#*, *http://xmlns.com/foaf/0.1/* e *http://sales.com/schema/*.

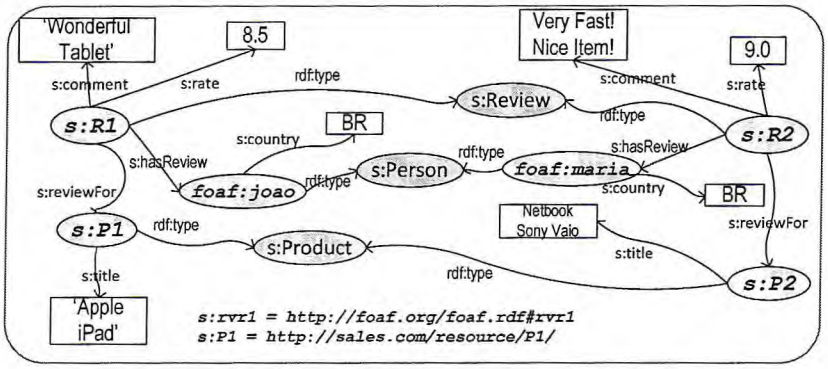


Figura 8 – Exemplo de um grafo RDF

2.4 Esquema RDF

Além do *modelo de dados*, há o esquema RDF, prefixo RDFS– *Resource Description Framework Schema* (BRICKLEY e GUHA, 2004) com a qual podem ser definidos conceitos e taxonomias, assim criando uma ontologia mais rica.

Um esquema (ou vocabulário) RDF fornece um conjunto de primitivas, das quais se destacam as que permitem definir classes, hierarquia de classes e propriedades com domínio e contradomínio. O domínio refere-se à classe onde é aplicada a propriedade, o contradomínio denota os valores possíveis para a propriedade.

A Figura 9 descreve uma taxonomia entre classes que demonstra o uso das primitivas referidas e esclarece a aplicabilidade das primitivas *domain* e *range* em propriedades.

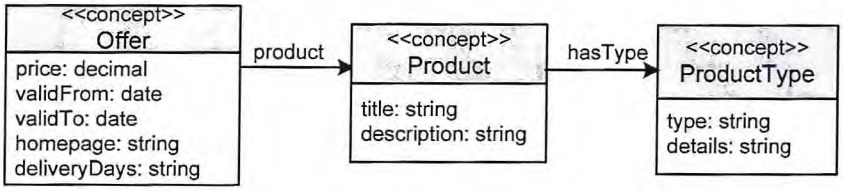


Figura 9 – Exemplo de um Esquema RDF

Por exemplo, *Offer* é declarado como classe; *title* é definido como propriedade com domínio *Product* e contradomínio (*range*) *string;* e *s:hasType* é definido como propriedade de objeto com domínio *Product* e contradomínio *ProductType*.

2.5 Linguagem OWL

Existem várias linguagens para codificação de uma ontologia, contudo, no presente trabalho, foca-se na *Web Ontology Language* (OWL) de Bechhofer et al. (2004), recomendação do W3C para o desenvolvimento e manipulação de ontologias na Web baseada em RDF e esquema RDF utilizando sintaxe XML.

OWL descreve as classes e propriedades e seus relacionamentos de maneira que facilita a interpretação automática de conteúdo da Web, que pode ser serializada em um conjunto de triplas RDF. Um esquema OWL é formado por classes com propriedades, definição de instâncias, declaração de propriedades e interpretação do relacionamento lógico entre as classes e instâncias permitidas pela semântica formal. A família linguagem OWL é organizada em três sublinguagens de acordo com a capacidade de expressão (BECHHOFER et al., 2004): OWL *Lite*, OWL DL e OWL *Full*. Cada tipo se adéqua às necessidades de grupos específicos de usuários e desenvolvedores de ontologias. A seguir detalha-se cada tipo.

OWL/Lite – herda do RDF um esquema de mecanismos de definição de classes e propriedades, de hierarquização de classes e documentação da especificação esquema RDF. Além disso, o módulo OWL/Lite fornece a semântica necessária para definir características de propriedades (transitividade, simetria, relação inversa), igualdade e desigualdade entre classes, propriedades e instâncias, tipos de dado das propriedades segundo a especificação de um esquema XML e restrições na cardinalidade máxima e mínima de propriedades (0 ou 1).

OWL/DL – permite explorar os mecanismos de raciocínio em lógica descritiva existente e é computável. Fornece suporte a propriedades que permitem definir relações entre instâncias de classes. No módulo OWL/DL, classes podem ser construídas por união, intersecção e complemento ou pela enumeração de instâncias, além de poder ter disjunções.

OWL/Full – utiliza o mesmo vocabulário da OWL DL, porém com suporte completo

quanto à cardinalidade máxima e mínima de propriedades (números inteiros positivos arbitrários, por exemplo). Enquanto os outros módulos da OWL restringem a sintaxe de classes definidas por esquemas RDF e pequenas extensões, por exemplo, owl:equivalentClass – o módulo OWL/*Full* permite a criação e manipulação de metaclasses via enumerações, restrições de propriedades e combinações lógicas. No entanto, não há garantia de *computabilidade* desse vocabulário.

As linguagens menos expressivas estão contidas nas mais expressivas, de maneira que uma ontologia definida em linguagem menos expressiva é aceita integralmente por uma linguagem mais expressiva. Além disso, toda ontologia OWL (*Lite*, DL ou *Full*) é um documento RDF.

Neste trabalho, usa-se OWL/*Lite*, pois é suficiente para especificar as ontologias utilizadas no processo de integração de dados. Resumidamente, as ontologias suportam tipos de dado (*datatype*) chamados de classes, propriedades de objeto, subclasses e indivíduos. O domínio de um tipo de dado ou propriedade de objeto é uma classe, o contradomínio de uma propriedade de tipo de dados é um tipo definido em conformidade com o esquema XML, enquanto o contradomínio de uma propriedade de objeto é uma classe. Os *InverseFunctionalProperty*, que se assemelham à noção de uma chave simples, *em banco de dados, para propriedades de objeto.*

2.6 Linked Data

O formato XML tem sido amplamente utilizado para publicação de dados na Web (VIDAL et al, 2004) (FERNANDEZ et al, 2001), contudo, a principal limitação reside na dificuldade para integração, em que os dados não estão ligados entre si como acontece com a Web de documentos. Os *Linked Data* vêm preencher essa lacuna.

Linked Data é um padrão da Web Semântica recomendado pelo W3C que se baseia na representação de dados em forma de triplas RDF e se refere à utilização da web para interligar dados relacionados (BIZER; HEATH; BERNERS-LEE, 2009).

De acordo com Berners-Lee et al, (2006), o termo *Linked Data* refere-se a um estilo de publicar e interligar dados estruturados na Web, em que o principal objetivo é permitir o compartilhamento de dados estruturados de forma tão fácil quanto documentos HTML são compartilhados atualmente.

Essa tecnologia permite muitas possibilidades para a integração de dados, pois lida com dados em escala global. Em geral, os dados no padrão de *Linked Data* são (BIZER; HEATH; BERNERS-LEE, 2009):

a) abertos – pode-se acessar *Linked Data* por meio de uma variedade ilimitada de aplicações e aplicativos, pois ela é expressa em formatos abertos, não-proprietários;
b) modulares – os *Linked Data* podem ser combinados com quaisquer outros *Linked Data*. Nenhum planejamento prévio é necessário para integrar essas fontes de dados, já que ambas utilizam o mesmo padrão;
c) escaláveis – é fácil adicionar mais *Linked Data* aos já existentes, mesmo quando os termos e definições utilizadas mudem ao longo do tempo.

Os *Linked Data* apóia-se em um pequeno conjunto de padrões já bem estabelecidos e amplamente utilizados na Web: um mecanismo de identificação global e único (URIs - *Uniform Resource Identifiers*), um mecanismo de acesso universal (HTTP - *Hypertext Transfer Protocol*), o modelo de dados RDF, e a linguagem de consulta SPARQL para acesso aos dados.

URIs (BERNERS-LEE; FIELDING; MASINTER, 2005) são usadas no contexto de *Linked Data* para identificar objetos e conceitos, permitindo que eles sejam desreferenciados para obtenção de informações a seu respeito. Assim, uma URI desreferenciada resulta em uma descrição RDF do recurso identificado. Por exemplo, a URI *http://www.w3.org/People/Berners-Lee/card#i* identifica o pesquisador *Tim Bernes-Lee* (MAGALHAES et al., 2011).

Nesse contexto, o modelo de dados RDF traz uma série de benefícios, como a criação de URI-*links* (links RDF) entre dados de diferentes fontes de dados, os quais permitem que informações de diferentes fontes se mesclem naturalmente e permitem a representação de informações de diferentes esquemas em um modelo único.

Neste trabalho, as fontes de dados estão no padrão de *Linked Data* como conjuntos de triplas RDF. A sua definição é descrita a seguir:

Definição 3 (Fonte de dados no padrão de *Linked Data*) – uma fonte de dados F é um conjunto de triplas RDF $(s, p, o) \in T^F$. Sendo que $s \in U$, $p \in U$ e $o \in U$, *isto é,* identificados unicamente por uma URI (U) que pode ser desreferenciada. □

Ademais, existem diversas maneiras de armazenar dados no padrão de *Linked Data*, por exemplo: usando RDF nativo por meio de APIs (*Application Programming Interface*) como *Sesame*[2], *Jena* (CARROLL et al, 2004), ou fornecendo *wrappers* para banco de dados relacionais, como D2R (BIZER e CYGANIAK, 2006), *Virtuoso*[3], também, pode-se previamente fazer a *triplificação* de fontes de dados relacionais usando, por exemplo, o Jena SDB[4] ou Jena TDB[5]. Além disso, esses dados no padrão de *Linked Data* podem ser acessados através de consultas SPARQL submetidas aos SPARQL *endpoints*, que serão introduzidos nas seções 2.7 e 2.8 respectivamente.

2.7 SPARQL

SPARQL (*SPARQL Protocol and RDF Query Language*) é uma recomendação W3C desde janeiro de 2008, que pode ser dividida em três partes: uma linguagem de consulta declarativa (principal componente SPARQL), um protocolo de acesso a dados em RDF (CLARK; FEIGENBAUM; TORRES, 2008) e um formato de resultado (BECKETT; BROEKSTRA, 2006).

O uso desses três padrões, que serão apresentados nas próximas subseções, contorna a heterogeneidade sintática entre diferentes fontes de dados, permitindo que sejam acessadas de forma consistente.

2.7.1 Linguagem de consulta SPARQL

SPARQL é uma linguagem de consulta declarativa, recomendada pelo W3C para extrair informações de grafos RDF. Sua arquitetura em relação aos padrões da web semântica é apresentada na Figura 10. Em suma permite extrair dados de grafos RDF com base em correspondência de padrões de tripla (ARENAS; PEREZ, 2011), que é o bloco de construção básica, definidos a seguir.

---

[2] http://www.openrdf.org/
[3] http://virtuoso.openlinksw.com/
[4] http://jena.hpl.hp.com/wiki/SDB
[5] http://www.openjena.org/TDB/

Figura 10 – Web Semântica e *Linked Data*

Definição 4 (padrão de tripla) – sendo *V* um conjunto de variáveis {*?v1*, *?v2*, *?v3*, ...}. Um padrão de tripla é uma tripla da forma $tp \in (U \sqcup B \sqcup V) \times (U \sqcup V) \times (U \sqcup B \sqcup L \sqcup V)$. □

Um exemplo de um padrão de tripla é *(?product, s:title, ,?t)*.

Definição 5 (padrão de grafo) – um padrão de grafo é um conjunto de padrões de triplas que adicionam restrições nas consultas SPARQL. Ela é definida pela gramática BNF (*Backus–Naur Form*) seguinte:

gp → *tp* | *gp AND gp* | *gp OPT gp* | *gp UNION gp* | *gp FILTER expr*

Sendo que *AND*, *OPT*, e *UNION* são operadores binários que correspondem aos operadores SPARQL conjunção ".", *OPTIONAL* e *UNION*, respectivamente. *FILTER expr* representa o construtor *FILTER* com uma expressão lógica (*expr*), que é formada por elementos do conjunto *UVL* (*URIs*, Variáveis e Literais), constantes, conectivos lógicos (¬, Λ, V), símbolos de desigualdade (<, <=, >, >=), símbolo de igualdade (=), predicados unários como *bind*, *isURI* e outros recursos definidos em Prud'hommeaux e Seaborne (2008). □

Definição 6 (consulta SPARQL) – uma consulta SPARQL é definida como: ***SELECT*** *varlist* ***WHERE*** *(gp)*, onde *varlist=(v1; v2; ...;vn)* é uma lista ordenada de variáveis e $varlist \subseteq var(gp)$; a função *var(gp)* retorna o conjunto de variáveis que ocorrem em *gp*. □

As cláusulas *SELECT* e *WHERE* são obrigatórias. Sendo que a cláusula *SELECT* projeta os resultados que serão retornados e a cláusula *WHERE* determina os padrões de triplas/grafos a ser localizados na fonte de dados RDF. As condições passíveis de utilização na cláusula *WHERE* do SPARQL diferem bastante do SQL, em que são introduzidas variáveis em substituição do sujeito, predicado ou objeto. As variáveis são prefixadas com um ponto de interrogação.

Considere a seguinte consulta sobre uma ontologia *reviewer.owl* apresentada na Figura 11, que recupera o nome e o email dos revisores localizados nos Estados Unidos.

```
PREFIX ps: <http://sales.org/reviewer/>
SELECT ?name, ?email
FROM <reviewer.owl>
WHERE {
 ?rv r:name ?name .
 OPTIONAL { ?rv r:email ?email }
 ?rv r:country ?country .
 FILTER regex(?country, "USA")
}
```

Figura 11 – Exemplo de uma consulta SPARQL

A cláusula *prefix* é uma expressão SPARQL usada para declarar o esquema, sendo bastante útil para simplificar a formulação das consultas, pois ajuda a distinguir os termos existentes por meio de espaço de nomes. A declaração do prefixo *r* permite ter *r:email* em vez de *<http:// sales.org/reviewer#email>*, simplificando a expressão SPARQL.

As variáveis *?name* e *?email* são projetadas na cláusula *SELECT*. A cláusula *FROM* identifica que fontes de dados devem ser consultadas. E a cláusula *WHERE* é um grafo composto por uma conjunção de triplas.

Quando padrões de tripla contêm variáveis comuns, precisam fazer a junção dos resultados de dois padrões de tripla, o operador *AND* (".") denota essa operação. O operador *OPTIONAL* torna uma parte do padrão de tripla opcional. A cláusula *FILTER* restringe os revisores dos Estados Unidos.

A semântica e a complexidade da linguagem de consulta SPARQL têm sido amplamente estudadas, mostrando que a álgebra SPARQL tem a mesma expressividade da álgebra relacional (PÉREZ; ARENAS; GUTIERREZ, 2009; BUIL-ARANDA; ARENAS; CORCHO, 2011), embora a conversão não seja trivial (ANGLES; GUTIERREZ, 2008).

É importante frisar que as operações de junção são o principal gargalo para execução de consulta SPARQL. Uma quantidade significativa de esforços de pesquisa têm se dedicado para processamento de operações de junção em banco de dados relacionais, e muitos algoritmos eficientes têm sido desenvolvidos, por exemplo, *index join* e *merge-join.*

Trabalhos atuais (STOCKER et al, 2008; NEUMANN; WEIKUM, 2010) exploraram a otimização de consultas SPARQL (GROPPE et al, 2009). Algumas estratégias são baseadas em heurísticas que incluem reordenação de padrões de tripla com base em estimativas de seletividade (STOCKER et al, 2008). Uma tendência recente é o desenvolvimento de repositórios RDF altamente escalável. Implementação como *RDF3X* (NEUMANN; WEIKUM, 2010) e *BitMatrix* (ATRE et al, 2010) foca em estruturas de índice eficiente e ordenação de junção que permitem que consultas SPARQL sejam feitas a partir dos dados indexados.

Está fora do escopo deste trabalho uma etapa de otimização das subconsultas SPARQL enviadas para execução nos SPARQL *endpoints* (otimização local), ficando a critério dos sistemas que armazenam os *linked datas* a realização desta otimização.

2.7.2 Consultas SPARQL em várias ontologias

Visto que uma fonte de dados pode conter várias ontologias, é importante definir as ontologias que se pretendem consultar, sendo isso possível por meio da cláusula *FROM NAMED* (CARROLL et al, 2005), pois permitem a uma mesma consulta buscar dados em mais de um grafo RDF, oferecendo meios para agrupar um conjunto de fontes de dados em um único grafo e, então, se referem a esse grafo usando um URI.

Por exemplo, a consulta SPARQL, na Figura 12, solicita o título e o nome das editoras e também telefone em duas ontologias OWL (*amazon* e *publishers*), com a cláusula *FROM NAMED*. Essas ontologias serão apresentadas na Figura 29.

```
PREFIX a: <http://example.org/amazon/>
PREFIX pub: <http://example.org/publishers/>
SELECT distinct ?title ?name ?phone
 FROM NAMED <amazon.owl>
 FROM NAMED <publishers.owl>
WHERE {
 GRAPH <amazon.owl> {
 ?p a:title ?title .
 ?p a:hasPub ?pb .
 ?pb a:name ?name .
 } .
 GRAPH <publishers.owl> {
 ?pub pu:name ?name .
 ?pub pu:phone ?phone .
} }
```

Figura 12 – Consulta SPARQL usando *named graph*

2.7.3 Protocolo SPARQL e formato dos resultados

Ao contrário do SQL, a linguagem SPARQL inclui um protocolo de acesso para RDF (CLARK; FEIGENBAUM; TORRES, 2008) associada, uma recomendação da W3C, em que consultas SPARQL podem ser executadas com chamadas SOAP (*Simple Object Access Protocol*) ou HTTP (*Hypertext Transfer Protocol*) (vide Figura 14 e Figura 13). O objetivo do protocolo SPARQL é promover a interoperabilidade, em que os clientes podem interagir com os dados SPARQL de forma consistente. Isso inclui a serialização dos resultados por meio de

streams HTTP, desse modo, possibilitando uniformizar todo acesso às fontes de dados independentemente da tecnologia utilizada, pois só é preciso executar os métodos HTTP *GET* ou *POST*.

```
POST /services/sparql-query HTTP/1.1
Content-Type: application/soap+xml
Accept: application/soap+xml
User-Agent: Axis/1.2.1
Host: debpedia.org
SOAPAction: ""
Content-Length: 438

<?xml version="1.0" encoding="UTF-8"?>
 <soapenv:Envelope
 xmlns:soapenv = "http://www.w3.org/soap-envelope/"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soapenv:Body>
 <query-request xmlns="http://www.w3.org/sparql-protocol-types/#">
 <query>SELECT ?z WHERE{?x ?y ?z ...}</query>
 </query-request>
 </soapenv:Body>
 </soapenv:Envelope>
```

Figura 13 – Exemplo do protocol SPARQL com SOAP

```
POST /sparql/?query=<encoded query> HTTP/1.1
Host: dbpedia.org
User-agent: my-sparql-client/0.1
```

Figura 14 – Exemplo do protocol SPARQL com HTTP

Para formatação dos resultados da consulta, existe outra recomendação da W3C (*SPARQL Query Results XML Format*) que padroniza a codificação de resultados de uma consulta SPARQL em XML (BECKETT; BROEKSTRA, 2006).

Ambas as recomendações se complementam para promover a interoperabilidade, permitindo interação com as fontes de dados, sem precisar tratar programaticamente com as heterogeneidades sintáticas entre diferentes fontes de dados. Essa abordagem é utilizada por muitos repositórios, incluindo *DBpedia*[6] – uma extração de informações estruturadas da Wikipédia e DBLP[7] –um banco de dados de publicações científica em ciência da computação.

2.8 SPARQL *endpoint*

Na forma mais simples, um SPARQL *endpoint* é um serviço de consulta baseado no protocolo HTTP que adere ao protocolo SPARQL, conforme definido pela recomendação

[6] http://dbpedia.org/
[7] http://www4.wiwiss.fu-berlin.de/dblp/

do W3C, em que consultas podem ser enviadas ao SPARQL *endpoint* e respondida apropriadamente.

Eles têm sido grande aliado para o sucesso e disseminação da utilização da linguagem SPARQL, pois possibilitam a execução de consultas de dados disponíveis no padrão de *Linked Data* na Web. Alguns exemplos são *DBpedia*[8], *Data.Gov*[9], *DrugBank*[10], dentre outros.

Com o advento da SPARQL 1.1 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011), a utilização de SPARQL *endpoint* torna-se ainda mais geral, uma vez que é possível enviar instruções SPARQL de atualização (*insert*, *update*, *delete*), usar operação HTTP em uma maneira *RESTful,* bem como executar consultas federadas, como apresentado na próxima seção.

2.9 Consultas SPARQL federadas

O processamento de consultas federadas sobre bancos de dados heterogêneos tem sido tema pesquisado há bastante tempo (SHETH; LARSON, 1990; KOSSMANN, 2000).

Consultas federadas sobre SPARQL *endpoints* permitem que os dados possam ser integrados, sendo potencialmente benéficas em fontes de dados heterogêneos cujos componentes individuais possam usar *wrappers* SPARQL para publicar dados. E a integração de dados pode se beneficiar da execução de consultas SPARQL federadas e é particularmente atraente devido aos progressos realizados pelo *Linked Data Project*[11], que visa promover a utilização generalizada das representações baseadas em URI. O uso consistente de URIs para representar os mesmos termos em diferentes fontes de dados permite que a junção de dados seja possível, portanto as consultas federadas podem fornecer resultados que não são obtidos a partir de uma única fonte de dados.

Vale destacar que, atualmente, a recomendação SPARQL só permite executar consultas SPARQL em *endpoints* isolados. Existe um grupo de trabalho do W3C especificando uma nova versão do SPARQL (*SPARQL 1.1 working draft*) (PRUD'HOMMEAUX; BUIL-ARANDA, 2011) que se concentra principalmente na

---

[8] http://dbpedia.org/sparql
[9] http://services.data.gov.uk/finance/sparql
[10] http://www4.wiwiss.fu-berlin.de/drugbank/sparql
[11] *Linked Data* – Connect Distributed Data across the Web (http://linkeddata.org/)

descrição de uma extensão para execução de consultas federadas e inclui dois novos operadores na linguagem: *SERVICE* e *BINDINGS* (PRUD'HOMMEAUX; BUIL-ARANDA, 2011). O operador *SERVICE* permite especificar, dentro de uma consulta SPARQL, o *endpoint* (serviço de consulta SPARQL) em que uma parte da consulta será executada. O operador *BINDINGS* permite a transferência de resultados para restringir uma consulta. Buil-Aranda et al. (2011) apresentam uma formalização da sintaxe e semântica de uma extensão que possibilita a execução de consultas federadas no SPARQL 1.1.

2.10 Considerações do capítulo

Neste capítulo, apresentou-se um panorama geral dos principais padrões utilizados na Web Semântica que possibilitam a disseminação dos *Linked Data.* Os padrões incluem RDF/RDFS, OWL e SPARQL, os quais alguns conceitos e definições foram também introduzidos no decorrer do capítulo. Finalmente, foi abordado o processamento de consultas SPARQL federadas, os SPARQL *endpoints*, solução adotada na implementação de integração de dados no padrão de *Linked Data.*

Também foram definidos termos e conceitos utilizados neste trabalho. Em especial as propriedades *URI-links* e ligações *same-as.* Isto mostra-se como um diferencial deste trabalho, pois possibilitam que os resultados de consultas sejam mais significativos e completos para o usuário, conforme será discutido no Capítulo 7.

3 INTEGRAÇÃO DE DADOS BASEADA EM ONTOLOGIAS

Neste capítulo são apresentados os conceitos relacionados à integração de dados relevantes neste trabalho. Na seção 3.2, apresenta-se uma visão geral sobre as abordagens para integração de dados. Na seção 3.3 discute-se a arquitetura de mediadores para integração de dados. Na seção 3.4 explica-se o uso de ontologias para integração de dados. Após isso, na seção 3.5, mostram-se os enfoques para definir mapeamentos entre ontologias que podem ser homogêneos ou heterogêneos.

3.1 Introdução

O problema de integração de dados tem sido tema de pesquisa na área de banco de dados há bastante tempo (ZIEGLER; DITTRICH, 2004) sendo reconhecido como amplo e de grande relevância. Um dos destaques das pesquisas é a proposição de uma arquitetura conceitual de integração de dados baseada em mediadores. De acordo com Lenzerini (2002), Halevy, Rajaraman e Ordille (2006), os principais componentes de um sistema de integração de dados são: o esquema de mediação; os esquemas das fontes de dados, em que os dados estão efetivamente armazenados; e os mapeamentos, que especificam as correspondências entre os esquemas das fontes de dados e o esquema de mediação.

Destaca-se como principal objetivo dos sistemas de integração de dados permitir que usuários consultem simultaneamente múltiplas fontes de dados heterogêneas e distribuídas e autônomas por meio de uma única interface de consultas, mantendo transparentes os procedimentos de acesso, extração e integração dos dados. Assim o sistema de integração de dados deve tratar de forma transparente para o usuário problemas de heterogeneidade (estrutural, conceitual e tecnológica) e distribuição e autonomia das fontes durante a execução de consultas.

De acordo com Özsu e Valduriez (1999), os principais desafios para a integração de dados são justamente esses três aspectos ortogonais: heterogeneidade, distribuição e autonomia, descritos a seguir.

a) Distribuição – as fontes de dados estão dispersas geograficamente, sendo interligadas por meio de uma rede de computadores. Como consequência da distribuição, é necessário lidar com os problemas envolvidos nas redes, como replicação, fragmentação e

custo da transmissão dos dados e a capacidade de processamento de cada servidor (SPARQL *endpoint*). Neste trabalho, várias fontes de dados dos mais diversos domínios estão publicadas na Web no padrão de *Linked Data* e usam *URI-Links* que possibilita acesso a dados localizados em outras fontes de dados.

b) Autonomia – refere-se ao nível de independência de operação de cada fonte de dados que participe de um sistema de integração, em que as fontes possuem controle total sobre os dados e, geralmente, não podem afetar suas funcionalidades e requerer modificações. Neste trabalho, os SPARQL *endpoints*, são públicos e acessados livremente por qualquer usuário ou aplicação na Web com uma simples requisição HTTP GET ou POST.

c) Heterogeneidade – como os esquemas das fontes são desenvolvidos independentemente, eles possuem estruturas e terminologias diferentes (heterogeneidade estrutural e semântica), o que ocorre tanto com os esquemas que vêm de domínios diferentes, quanto com os modelados no mesmo domínio do mundo real, pelo fato de ser desenvolvidos por pessoas diferentes, em diferentes contextos. Para serem efetivos, os sistemas de integração de dados devem ser capazes de transformar dados de diferentes fontes para responder a consultas feitas sobre esse esquema. Uma solução para minimizar os problemas de heterogeneidade consiste na adição de mais semântica aos metadados (WACHE et al., 2001; LEHTI; FANKHAUSER, 2004). Embora sejam basicamente independentes um do outro, existem correlações. Por exemplo, quanto maior a autonomia de duas ou mais fontes de dados, maior é, geralmente, o nível de distribuição e heterogeneidade. Neste trabalho, utiliza-se uma ontologia denominada ontologia de domínio para definir um vocabulário padrão que abstrai o vocabulário heterogêneo das fontes de dados.

3.2 Abordagens para integração de dados

Os sistemas de integração de dados seguem, normalmente, duas abordagens clássicas segundo Lenzerini (2002): virtual ou materializada. A abordagem materializada extrai os dados de fontes distintas e materializa-os localmente em repositórios chamados *datawarehouses*. As consultas são realizadas sobre a base materializada e, dessa forma, apresentam melhor desempenho em relação à abordagem virtual. A desvantagem dessa abordagem é a necessidade de manter a base materializada sempre atualizada. Em alguns

casos, é necessário a presença de um administrador para isso, caso contrário poderão ser geradas respostas com informações desatualizadas.

Na abordagem virtual, os dados são recuperados diretamente das fontes quando o sistema de integração precisa responder a uma consulta, ou seja, sistemas que seguem essa abordagem enviam consultas diretamente às fontes de dados e, após os resultados individuais obtidos, integram os dados e fornecem o resultado final ao usuário ou à aplicação que os requisitou. Os dados acessados por essa abordagem estão sempre atualizados, entretanto, sabe-se que os custos de processamento das consultas e de acesso às fontes são fatores consideráveis na escolha.

A escolha da abordagem a ser seguida está diretamente relacionada à arquitetura e às características da aplicação de integração. Em alguns casos, as duas abordagens podem até ser utilizadas concomitantemente dentro de uma mesma solução.

Neste trabalho adota-se a abordagem virtual, devido ao fato de que os *Linked Data* estarem disponíveis em escala global e serem atualizados livremente.

3.3 Utilização de mapeamentos nas tarefas de integração

Para que ocorra a integração de dados, devem ser definidos relacionamentos ou mapeamentos entre cada esquema fonte e o esquema de mediação. Um mapeamento especifica como instâncias de dados de um esquema correspondem à instância de dados de outro esquema. Mapeamentos são freqüentemente especificados de forma declarativa e independente de dados (YU; POPA, 2004).

3.3.1 Abordagem para definição de mapeamentos

Seja qual for a abordagem adotada, é fundamental para os sistemas de integração que se estabeleça como os elementos do esquema de mediação estão relacionados com os elementos dos esquemas locais. Portanto, são definidos os mapeamentos entre o esquema de mediação e os esquemas das fontes de dados, as quais possibilitam tanto materializar o esquema de mediação (abordagem materializada) quanto reescrever consultas sobre o esquema de mediação em subconsultas sobre as fontes locais (abordagem virtual). Os enfoques de definição de mapeamentos mais comumente encontrados na literatura

(LENZERINI, 2002) são: a) *Global-As-View* (GAV), em que o esquema de mediação é definido pelos mapeamentos como uma visão sobre os esquemas locais; e b) *Local-As-View* (LAV), em que os mapeamentos definem os esquemas das fontes locais como visões sobre o esquema de mediação.

Uma arquitetura convencional de sistemas de integração que segue a abordagem virtual é baseada em mediadores (WIEDERHOLD, 1992). Neste trabalho, adota-se a arquitetura de mediadores, apresentada na próxima seção, que segue a abordagem GAV para definição dos mapeamentos denominados de mapeamentos de mediação. Esta abordagem favorece a reformulação de consulta, pois uma consulta submetida a um esquema de mediação pode ser simplesmente desdobrada e expressa em termos de subconsultas sobre as fontes de dados.

3.3.2 Reformulação de consulta

Um sistema de integração de dados deve possibilitar a reformulação de consultas, independentemente do método utilizado para especificação dos mapeamentos (GAV ou LAV). O objetivo é responder consultas formuladas sobre um esquema alvo (esquema de mediação), a partir de mapeamentos previamente definidos entre o esquema de mediação e as fontes de dados.

O problema da reformulação de consultas pode ser dividido em duas etapas (HALEVY, 2000): a) reescrita da consulta: consiste na tradução de uma consulta em uma expressão de consulta, formada por uma ou mais subconsultas; b) resolução da consulta: consiste em realizar a combinação dos resultados das subconsulta em várias fontes de dados e em retornar um resultado coerente ao usuário.

A saída de um algoritmo de reformulação é o plano de execução da consulta Q' usado para responder a consulta Q. Um ponto importante a salientar na reformulação de consultas é relativo à correção e completude de algoritmos (HALEVY, 2000). Dada uma consulta Q e um conjunto de esquema das fontes de dados F_1, F_2, ..., F_n, esse algoritmo sempre é capaz de achar uma consulta Q' sobre as fontes de dados; caso a consulta exista, esse algoritmo é considerado completo. Outro ponto a ser observado é se a consulta Q' está correta, ou seja, se ela gera apenas tuplas válidas como resultado, caso em que o algoritmo é considerado correto.

Um exemplo de algoritmo de reformulação de consulta completo foi proposto em (YU; POPA, 2004). Esta abordagem permite definir a semântica da resposta de uma consulta sobre um esquema de mediação, considerando um conjunto de instâncias das fontes de dados e os mapeamentos entre o esquema de mediação e os esquemas-fonte. Este algoritmo permite recuperar todas as respostas de acordo com a semântica dada pelas instâncias do esquema de mediação. Assim sendo, a união dos dados consultados nas fontes tem o mesmo efeito que os dados recuperados pela execução direta da consulta sobre o esquema de mediação.

3.4 Uso de mediadores para integração de dados

Os mediadores são os responsáveis por disponibilizar o esquema de global (também chamado de esquema de mediação). Os usuários desses sistemas submetem as consultas a um esquema de mediação (WIEDERHOLD, 1992), necessitando conhecer apenas a estrutura do esquema e a linguagem de consulta adotada.

Ao receber uma consulta sobre o esquema, o mediador a reescreve em subconsultas que serão executadas pelas diversas fontes de dados e integrar os resultados obtidos. Como as fontes podem ser baseadas em tecnologias, formatos e modelos completamente distintos, os mediadores normalmente se utilizam de tradutores (*wrappers*) para o acesso e execução das consultas sobre as fontes de dados. Após obter os resultados locais, os mediadores consolidam o resultado e retornam ao usuário a resposta da consulta realizada sobre o esquema de mediação.

Na arquitetura de mediadores, módulos tradutores são responsáveis pela exportação de visões que representam as diversas fontes de dados conforme um modelo de representação comum. Na camada de mediação, essas visões são integradas para formar um esquema de mediação sobre o qual são submetidas às consultas dos usuários.

A camada de mediação do sistema pode conter vários mediadores, que acessam e consultam diretamente as fontes de dados (por meio dos tradutores) ou até mesmo consultam outros mediadores, formando uma arquitetura multicamadas.

Na Figura 15 é exibida uma arquitetura com um único mediador, a qual também será utilizada neste trabalho. De forma resumida, os principais passos do processamento de consultas nesta arquitetura são os seguintes:

a) o usuário submete uma consulta sobre o esquema de mediação publicado pelo

mediador;

b) o mediador realiza a reescrita da consulta em subconsultas que serão enviadas aos tradutores das fontes de dados. Os *wrappers* executam as subconsultas sobre as fontes de dados e retornam os resultados para o mediador;

c) o mediador realizada a integração dos resultados das subconsultas e apresenta o resultado final ao usuário.

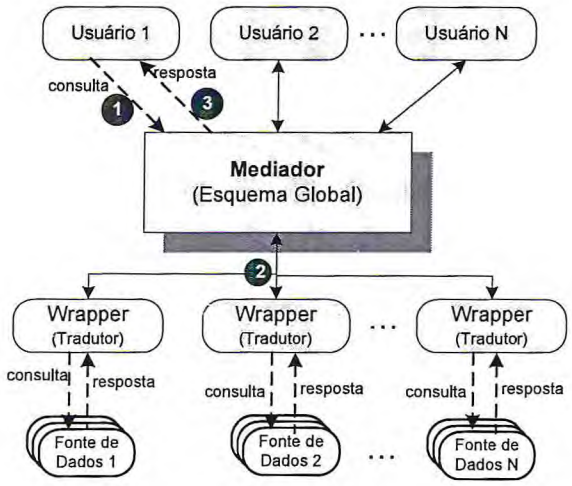


Figura 15 – Arquitetura de um sistema de integração baseada em mediação

3.5 Uso de ontologias para integração de dados baseada em mediador

Sistemas de integração de dados baseados em mediadores (LENZERINI, 2002; BARU et al., 1999), apresentados na seção anterior, foram propostos em 1992 (WIEDERHOLD, 1992). Inicialmente, esta arquitetura era baseada principalmente no modelo de dados relacional ou orientado a objetos. Uma vez que XML evoluiu como linguagem padrão para troca de dados na Web, alguns trabalhos recentes têm proposto enfoques baseados em XML para o projeto de sistemas de integração de dados (ESSID et al, 2004; MANOLESCU; FLORESCU; KOSSMANN, 2001; TEIXEIRA, 2009). Entretanto, a versatilidade do XML como modelo de dados e o poder de expressão das linguagens de consulta XML têm levado a uma arquitetura complexa para integração de dados, pois é

necessário tanto especificar a visão de mediação em XML quanto descobrir os mapeamentos entre a visão de mediação e os esquemas XML das fontes de dados. O que contribui significativamente para sua complexidade, que advém da estrutura aninhada e da flexibilidade na definição dos esquemas; por exemplo, dois documentos XML podem apresentar o mesmo esquema, mas árvores de estruturas diferentes. Essa complexidade se reflete, por exemplo, na ausência de uma álgebra de consenso para o processamento de consultas em documentos XML.

Nos últimos anos, o uso de ontologias em sistemas de integração de dados (MENA et al, 1996; CALVANESE et al., 2008; CRUZ; XIAO; HSU, 2004; LANGEGGER; WOSS; Blochl, 2008) tem se disseminado bastante para superar, principalmente, problemas causados pela heterogeneidade dos dados (HALEVY; RAJARAMAN; ORDILLE, 2006). As ontologias também têm sido usadas para diversos outros propósitos, incluindo: a) especificação de esquema de mediação (NOY, 2004), que possibilita uma visão conceitual sobre os esquemas da fonte heterogênea e um vocabulário compartilhado; b) representação de metadados, em que os metadados de cada fonte de dados são representados por uma *ontologia-fonte* (WACHE et al., 2001); c) suporte para consultas de alto nível, em que, dado um esquema de mediação, o usuário pode formular uma consulta sem conhecimento das especificidades das fontes de dados heterogêneas (AMANN et al., 2002). As ontologias possuem ainda regras de inferência que podem ser usadas para checar a consistência dos mapeamentos e inferir novos *conhecimentos* (NECIB, 2007).

Por outro lado, dados no padrão de *Linked Data* tornaram-se mais abundantes, heterogêneos e independentes devido, principalmente, à disponibilidade de SPARQL *endpoints* para acessá-los em escala global.

Para fornecer suporte à representação e processamento de dados no padrão de *Linked Data*, ontologias vêm sendo utilizadas como principal mecanismo para prover consistência e vocabulário uniforme de representação nesse ambiente aberto, heterogêneo e ubíquo. Em geral, existem três diferentes formas de realizar integração de dados usando-se ontologias (WACHE et al., 2001): os enfoques que utilizam ontologia global; os enfoques que utilizam múltiplas ontologias; e os enfoques híbridos.

A seguir, será visto cada um desses enfoques.

Ontologia global – os enfoques que utilizam uma única ontologia global fornecem um vocabulário compartilhado para a especificação da semântica de um domínio. Todas as fontes de dados se relacionam à ontologia global. Um modelo

independente de cada fonte de dados deve ser descrito pelo sistema, relacionando os objetos de cada uma ao esquema de mediação. Esse enfoque deve ser utilizado em problemas de integração quando todas as fontes de dados fornecerem praticamente a mesma visão do domínio num nível de granularidade semelhante. Na Figura 16, é fornecida uma visão geral do enfoque com uma única ontologia global.

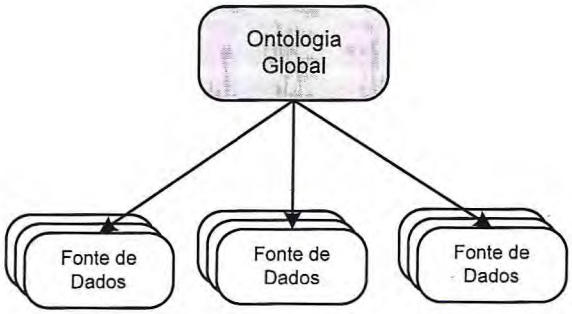


Figura 16 – Enfoque que utiliza uma única ontologia global
Fonte: adaptado de WACHE et al. (2001)

A desvantagem deste enfoque se dá quando ocorrem mudanças nas fontes de dados que podem afetar a conceitualização do domínio representado pela ontologia, resultando em mudanças na ontologia global e nos mapeamentos entre esta e as fontes de dados.

Múltiplas ontologias – no enfoque que utiliza múltiplas ontologias, cada fonte de dados é descrita pela própria ontologia-fonte, o que facilita as mudanças que porventura possam ocorrer nas fontes, como inclusão ou remoção de novas fontes de dados. Como desvantagem nessa abordagem, tem-se a ausência de um vocabulário compartilhado devido à dificuldade de realizar comparações entre ontologias diferentes. Ressalta-se que isto torna o processo de integração mais complexo. Para resolver o problema, um formalismo de mapeamento entre ontologias pode ser definido, a fim de identificar termos semanticamente correspondentes entre as ontologias-fonte. A Figura 17 fornece uma visão geral do enfoque com múltiplas ontologias.

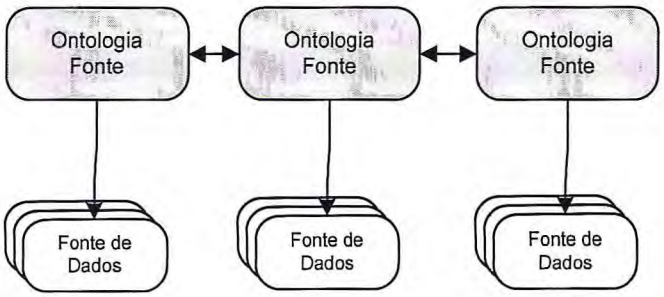


Figura 17 – Enfoque que utilizam múltiplas ontologias
Fonte: adaptado de WACHE et al. (2001)

Híbrido – por fim, tem-se o enfoque híbrido, que tenta conciliar as duas abordagens citadas anteriormente. Neste enfoque, a semântica de cada fonte é descrita pela própria ontologia-fonte, construída sobre um vocabulário compartilhado a fim de tornar possível a comparação entre as ontologias, na Figura 18 os relacionamentos entre as ontologias-fontes refletem esta possibilidade. Como cada termo de uma ontologia-fonte é baseado nos termos primitivos da fonte de dados correspondente, a comparação entre eles é mais fácil de ser realizada do que nos enfoques que utilizam múltiplas ontologias. A vantagem do enfoque híbrido é que suporta a aquisição e evolução de ontologias, o que é apropriado para lidar com dados no padrão de *Linked Data*. A desvantagem é que as ontologias preexistentes não podem ser reutilizadas com facilidade, pois todas as ontologias-fonte devem referir-se a um vocabulário comum. A Figura 18 ilustra a arquitetura dessa abordagem.

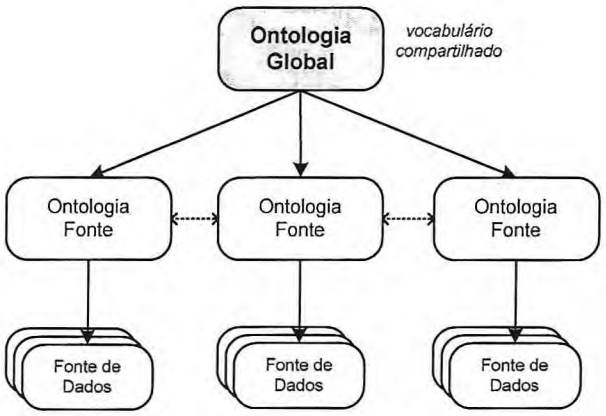


Figura 18 – Enfoque híbrido para Integração de Dados.
Fonte: adaptado de WACHE et al. (2001)

Os enfoques de ontologia global única e híbrido são adequadas para construção de sistemas de integração de dados, sendo a primeira ainda mais adequada a sistemas que utilizam a abordagem GAV e o enfoque de múltiplas ontologias são adequados para sistemas que fazem uso da abordagem LAV (CRUZ; XIAO, 2005).

Em suma, as ontologias podem ser muito úteis em projetos de integração de dados. Isso pode ser confirmado em função da quantidade de projetos e linhas de pesquisas que buscam integrar por intermédio dessa solução (CRUZ; XIAO; HSU, 2004; QUILITZ; LESER, 2008; LANGEGGER, 2010; PINHEIRO et al., 2010). Dessa forma, utilizar ontologias pode ser uma opção para lidar com a heterogeneidade dos vocabulários das fontes de dados no padrão de *Linked Data* e prover interoperabilidade.

3.6 Mapeamentos entre ontologias

Os mapeamentos entre ontologias podem ser homogêneos ou heterogêneos. Mapeamentos heterogêneos (GHIDINI; SERAFINI, 2006) expressam as relações semânticas entre duas ontologias, quando, por exemplo, a informação representada como uma classe na primeira ontologia é representada como uma propriedade de objeto (ligação) na segunda ontologia, e vice-versa. Com relação à direção, um mapeamento pode ser classificado como unidirecional ou bidirecional. Um mapeamento unidirecional especifica como se mapeiam elementos entre as duas ontologias e geralmente não é inversível. A regra de mapeamento bidirecional especifica como elementos de uma ontologia O_T mapeiam elementos de uma ontologia O_S, e vice-versa (SACRAMENTO et al., 2010).

Na literatura, duas arquiteturas para integração de dados baseada em ontologias podem ser identificadas para especificação dos mapeamentos entre ontologias, a saber: de dois níveis e de três níveis (CALVANESE et al., 2008; LANGEGGER; WOSS; BLOCHL, 2008; LUTZ, 2005; VIDAL et al., 2009).

Os principais componentes da arquitetura de dois níveis (Figura 19) são: a ontologia de domínio, que contém os termos essenciais de um domínio; as ontologias-fonte, que descrevem as fontes de dados usando uma linguagem de ontologia; e os mapeamentos, que especificam a correspondência entre as ontologias-fonte e a ontologia de domínio.

Os sistemas descritos em Calvanese et al. (2008) e Langegger, Woss e Blochl (2008) adotam a arquitetura de dois níveis.

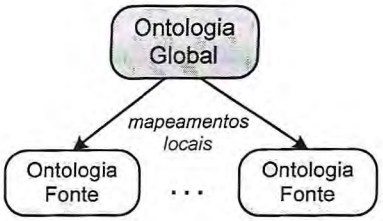


Figura 19 – Arquitetura de dois níveis baseada em ontologias
Fonte: adaptado de SACRAMENTO et al. (2010)

A arquitetura de três níveis (Figura 20) é composta pelos seguintes componentes: ontologia global; ontologias-fonte; ontologias de aplicação, que descrevem as ontologias-fonte usando um subconjunto do vocabulário da ontologia global; o mapeamento que especifica as correspondências entre as ontologias de aplicação e a ontologia global; e o mapeamento que especifica as correspondências entre as ontologias-fonte e as ontologias de aplicação (mapeamentos locais), segundo Sacramento et al. (2010).

Nesta arquitetura, as ontologias de aplicação são utilizadas para facilitar tarefas, como descoberta e recuperação de dados e também para integração de dados. O trabalho de Lutz (2005) adota essa arquitetura.

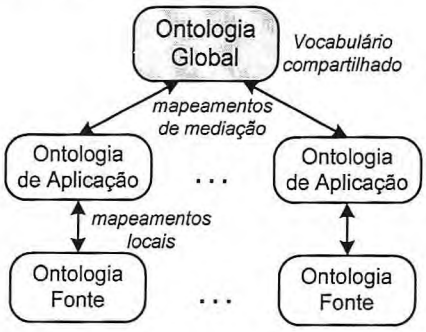


Figura 20 – Arquitetura de três níveis baseada em ontologias
Fonte: adaptado de SACRAMENTO et al. (2010)

Os principais problemas levantados por ambas as arquiteturas são: a) como especificar os mapeamentos entre ontologias; e b) como utilizar os mapeamentos para responder corretamente às consultas submetidas à ontologia de domínio.

Na arquitetura de dois níveis, a ontologia de domínio é usada apenas para especificar o esquema de mediação. Assim, para permitir a descoberta, recuperação e integração de dados, o usuário tem de definir os mapeamentos. Como as ontologias não compartilham o mesmo vocabulário e são estruturalmente heterogêneas, os mapeamentos entre as ontologias-fonte e as ontologias de aplicação podem ser denominados de mapeamentos heterogêneos (GHIDINI. SERAFINI, 2006).

Na arquitetura de três níveis, a ontologia de domínio é usada tanto para especificar o esquema de mediação como um vocabulário compartilhado. Visto que as ontologias de aplicação são fragmentos de uma ontologia de domínio, o usuário pode definir mapeamentos (chamados de mapeamento homogêneo) entre as duas ontologias.

Embora as arquiteturas de dois níveis e de três níveis sejam baseadas no *matching* (i.e. operação que permite a comparação de similaridade) entre os conceitos presentes nas ontologias-fonte e na ontologia de domínio, a arquitetura de três níveis acrescenta as ontologias de aplicação, que simplificam a definição dos mapeamentos de mediação, facilitando assim o processo de reformulação da consulta, bem como a integração de dados (SACRAMENTO et al., 2010).

As ontologias de aplicação são usadas para dividir a definição dos mapeamentos em duas etapas: mapeamentos de mediação e mapeamentos locais. Além disso, usam-se os mapeamentos de mediação para definir as classes e propriedades da ontologia de domínio em termos de classes e propriedades das ontologias de aplicação – utilizados para reescrever uma consulta submetida à ontologia de domínio em subconsultas expressa em termos das ontologias de aplicação. E os mapeamentos locais são utilizados para reescrever cada subconsulta sob as ontologias de aplicação em subconsultas sobre as ontologias-fonte.

Neste trabalho, se utiliza a arquitetura de três níveis, sendo que os mapeamentos de mediação são homogêneos, os mapeamentos locais são heterogêneos, sendo todos eles unidirecionais. A linguagem de mapeamento utilizada neste trabalho são correspondências entre conceitos e propriedades das ontologias. Os mapeamentos são relativamente simples e diretos, conforme apresentado no Capítulo 5.

3.7 Considerações do capítulo

Neste capítulo, foi apresentada uma visão geral dos sistemas de integração de dados abordando-se a arquitetura de mediadores. Mostrou-se a importância das ontologias em

projetos de integração de dados que pode ser confirmada em função da quantidade de projetos e linhas de pesquisas que buscam integração de dados por intermédio dessa solução. Dessa forma, utilizar ontologias pode ser uma opção para lidar com a complexidade dos ambientes atuais e prover interoperabilidade.

No *framework* que será apresentado no Capítulo 5, utiliza-se a arquitetura de três níveis com abordagem híbrida, semelhante à arquitetura apresentada na Figura 20, constituída também por ontologias de domínio e de aplicação. A ontologia de domínio fornece um vocabulário adequado constituído pelos termos básicos (vocabulário) de um domínio. Para cada fonte de dados no padrão de *Linked Data*, há uma ontologia de aplicação, descrita com o vocabulário comum da ontologia de domínio.

4 ESTADO DA ARTE

Afirmar que os *Linked Data* em escala global podem ser visto como um grande banco de dados induz algumas considerações sobre o uso de facilidades presentes no Sistema de Gerenciamento de Banco de Dados (SGBDs), como consultas declarativas, definição de um esquema comum, processamento eficiente de consultas, etc. Desse ponto de vista, é possível perceber que é preciso um longo caminho a ser percorrido para que os *Linked Data* se tornem, de fato, um vasto banco de dados. O ambiente Web dos *Linked Data* é pouco tolerante às técnicas intrusivas dos SGBDs, onde é possível alterar dados e estruturas, bem como a configuração dos ambientes de armazenamento e execução.

Por outro lado, soluções diferentes com objetivos semelhantes têm sido propostas na literatura para integração de dados (CRUZ et al., 2004; CALVANESE et al., 2008; QUILITZ; LESER, 2008; LANGEGGER, 2010; PINHEIRO et al., 2010). Estes trabalhos são baseados na abordagem virtual para integração de dados.

Não existe consenso na literatura sobre uma arquitetura de referência para a integração de dados em ambientes com fontes autônomas de dados distribuídos e heterogêneos. Independentemente da abordagem, arquitetura e tecnologia adotada, esses sistemas têm desafios comuns a serem tratados, como: a) linguagem de consulta; b) mapeamentos; c) reformulação de consulta; d) processamento eficiente de consultas.

A seguir, analisamos algumas propostas relevantes neste contexto. Ao final deste capítulo, é apresentada uma análise comparativa.

4.1 Trabalhos relacionados

Atualmente, os esforços estão sendo focalizados no uso de ontologias, principalmente para especificação de esquema de mediação (NOY, 2004), o qual possibilita uma visão conceitual sobre os esquemas de fontes heterogêneas e um vocabulário compartilhado. Por exemplo, Cruz et al. (2004) fornecem uma abordagem baseada em ontologias para a integração de documentos XML heterogêneos, transformando as fontes XML heterogêneas em ontologias-fonte RDF, que são combinadas em uma ontologia RDF

global. Este trabalho não define uma abordagem para a consolidação dos resultados obtidos a partir de fontes de dados diferentes e não explora o processamento eficiente de consultas.

Um extenso trabalho que trata o problema de reformulação de consulta em sistemas de integração de dados baseado em ontologias e lógica descritiva foi desenvolvido por Calvanese et al. (2007; 2008) que formalizaram um algoritmo para reformulação de consulta. O primeiro passo desse algoritmo é a reescrita da consulta (também chamada de expansão da consulta), que a partir de uma consulta Q computa nova consulta Q' sobre o esquema de mediação, em que os axiomas do esquema de mediação são compilados. Pode-se dizer que a consulta Q é reescrita de acordo com o conhecimento especificado no esquema de mediação que relevante para responder Q. Assim, a consulta Q é expandida de acordo com as restrições (axiomas) no esquema de mediação. A segunda etapa é chamada de *unfolding* da consulta, que usa os mapeamentos para traduzir a consulta expandida Q' em termos do esquema das fontes de dados, obtendo uma consulta Q''; e a terceira etapa *(execução da consulta)* executa a consulta Q'' nas fontes de dados e retorna o resultado final para o usuário.

Por outro lado, diversas abordagens de integração de dados para acessar fontes de dados RDF distribuídas utilizando SPARQL *endpoints* têm sido descritas na literatura. Dentre esses trabalhos se destacam: DARQ (QUILITZ; LESER, 2008) e SemWIQ (LANGEGGER, 2010), os quais esta proposta está fortemente relacionada, pois utilizam SPARQL como linguagem de consulta e protocolo de acesso (vide seções 4.3 e 4.4).

Hartig et al. (2009) propõem um modelo que explora a estrutura de navegação dos *Linked Data*. A ideia principal é descobrir dados que possam ser pertinentes para responder a uma consulta durante a execução. A descoberta é feita percorrendo as ligações (URIs *links*) entre as fontes de dados RDF; e os resultados parciais são armazenados num repositório local. Também é apresentado um algoritmo com iteradores baseado em *pipelining*. Embora essa abordagem seja promissora, as otimizações que estão sendo aplicadas são ainda bem simples e se concentram em explorar a natureza de navegação dos *Linked Data*.

Networked Graphs (SCHENK; STAAB, 2008) seguem uma abordagem baseada em visões de integração e fornece um formalismo que permite aos usuários definir grafos RDF usando visões sobre os outros grafos, possibilitando a composição de vários grafos que podem ser consultados de forma integrada. A composição é descrita de forma declarativa usando uma extensão da semântica do SPARQL. A implementação considera a aplicação do algoritmo de semi-junção (ZEMÁNEK; SCHENK, 2008). A desvantagem dessa solução é que requer extensões na linguagem de consulta SPARQL.

Também tem sido investigado o processamento de consultas federadas. O recente trabalho de Olaf e Staab (2011) destaca que o processamento eficiente de consultas SPARQL federadas ainda não é o ideal, e há espaço para melhorias. Neste mesmo trabalho foi apresentado o estado da arte para uma arquitetura de federação, que permite consultar transparentemente fontes distribuídas no padrão de *Linked Data* e, também, foi apresentada uma abordagem para otimização de consultas baseada em programação dinâmica. Além disso, os autores motivam o uso de várias técnicas e algoritmos consagrados de processamento de consultas distribuídas, especificamente a estratégia de semi-junção e a programação dinâmica foram explicadas em mais detalhes.

Nas seções seguintes, apresentam-se o sistema para processamento distribuído de consultas FeDeRate e o processador de consultas federadas ARQ/Service. Em seguida apresenta-se em mais detalhes o DARQ e o SemWIQ.

4.1.1 FeDeRate

Segundo Prud'hommeaux (2004), FeDeRate é um sistema para execução de consultas distribuídas sobre múltiplas fontes de dados cujo foco está em aplicações no domínio da bioinformática. As consultas são submetidas à federação em SPARQL e enviadas para as fontes de dados individuais, cujos resultados são combinados e retornados como o resultado da consulta federada. FeDeRate fornece suporte para consultas distribuídas com os padrões de grafos nomeados (*named graph patterns*) usando a sintaxe SPARQL para consultas de vários grafos nomeados. FeDeRate visa minimizar o tamanho dos resultados da consulta a fim de executá-las de forma mais eficiente, utilizando os resultados de consultas previamente executadas como valores constantes nas consultas subsequentes. Funcionalidades tais como reescrita de consulta e ordenação de junção não são contempladas. Assim, o FeDeRate pode ser visto como uma linguagem de banco de dados múltiplos, porque os *endpoints* têm de ser explicitamente especificados. Dessa forma, apresenta como desvantagens a necessidade do usuário em: a) conhecer a localização exata das fontes de dados; b) compreender detalhes dos esquemas dessas fontes de dados; e c) detectar e resolver conflitos causados por heterogeneidade de diversas naturezas.

4.1.2 ARQ/Service

O processador de consulta SPARQL ARQ (ARQ, 2008) é integrado ao framework Jena (CARROLL et al., 2004) para desenvolvimento de aplicações usando os padrões da Web semântica. Embora não seja uma solução abrangente de processamento distribuído de consulta, fornece extensões para executar consultas remotas, estendendo a linguagem SPARQL com o operador *SERVICE* que permite especificar a URI de um SPARQL *endpoint* em que parte de uma consulta será executada. Esse operador já faz parte da recente especificação SPARQL 1.1 Federated Query de Maio de 2011 (PRUD'HOMMEAUX; BUIL-ARANDA, 2011) que inclui também um novo operador denominado *BINDINGS* que permite, por exemplo, a transferência de resultados de uma execução anterior para restringir outra consulta que deve incluir o operador *BINDINGS*. Buil-Aranda, Arenas e Corcho, (2011) já definiram a sintaxe e a semântica desses operadores.

4.1.3 Distributed ARQ

O ARQ foi estendido pelo DARQ (QUILITZ; LESER, 2008), acrônimo de *Distributed ARQ*, um projeto do *HP Labs Bristol*. O principal objetivo é possibilitar a federação de consultas SPARQL, fornecendo acesso transparente a múltiplos SPARQL *endpoints*. Resumidamente, o DARQ decompõe uma consulta SPARQL em subconsultas e as submete aos SPARQL *endpoints* correspondentes, sendo os resultados obtidos de cada subconsulta integrados no mediador. Do ponto de vista arquitetural, DARQ é um mediador similar à arquitetura definida em Wiederhold (1992), vide Figura 21. No entanto, diferente de um sistema baseado em mediador, o DARQ não prevê a integração de esquemas.

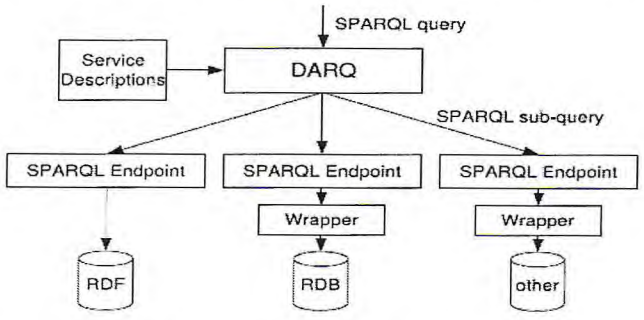


Figura 21 – Arquitetura de integração de dados do DARQ
Fonte: QUILITZ; LESER (2008)

Para usar o DARQ é necessário ter previamente armazenada a capacidade de cada fonte de dados em um arquivo chamado de descrição de serviço (*service descriptions*), um catálogo em RDF que descreve as fontes de dados. Essas capacidades são armazenadas localmente no mediador e descrevem os predicados presentes em cada fonte de dados com suas cardinalidades e também definem algumas informações de *seletividades* para expressões de filtro.

A Figura 22 mostra uma *descrição de serviço* para uma fonte de dados *FOAF*, que possui os predicados *foaf:name*, *foaf:mbox* e *foaf:weblog*. Objetos com predicado *foaf:name* começam com uma letra na faixa de *A* a *R* e possuem 52 triplas.

A fonte de dados possui um total de 112 triplas e tem limitações nos padrões de acesso, ou seja, uma consulta deve conter pelo menos um padrão de tripla com predicado *foaf:name* ou *foaf:mbox* com um objeto *bound*[12].

```
[] a sd:Service ;

      sd:capability [ sd:predicate foaf:name ;
       sd:objectFilter "REGEX(?object,"^[A-R]")" ;
       sd:triples 52 ] ;
      sd:capability[ sd:predicatefoaf:mbox ;
       sd:triples 50 ];
      sd:capability[sd:predicate foaf:weblog ;
       sd:triples 10 ];
      sd:totalTriples "112";
      sd:url "EndpointURL";
      sd:requiredBindings [ sd:objectBindingfoaf:name ] ;
      sd:requiredBindings [ sd:objectBindingfoaf:mbox ].
```

Figura 22 – Exemplo de descrição de serviço no DARQ
Fonte: adaptado de QUILITZ; LESER (2008)

A reformulação da consulta é bem simples (vide Figura 23): a consulta principal é decomposta (particionada) em várias subconsultas de acordo com as informações na descrição de serviço, cada uma das quais pode ser respondida por um SPARQL *endpoint* conhecido.

[12] Mais detalhes de descrições de serviço podem ser encontrados em http://darq.sf.net/.

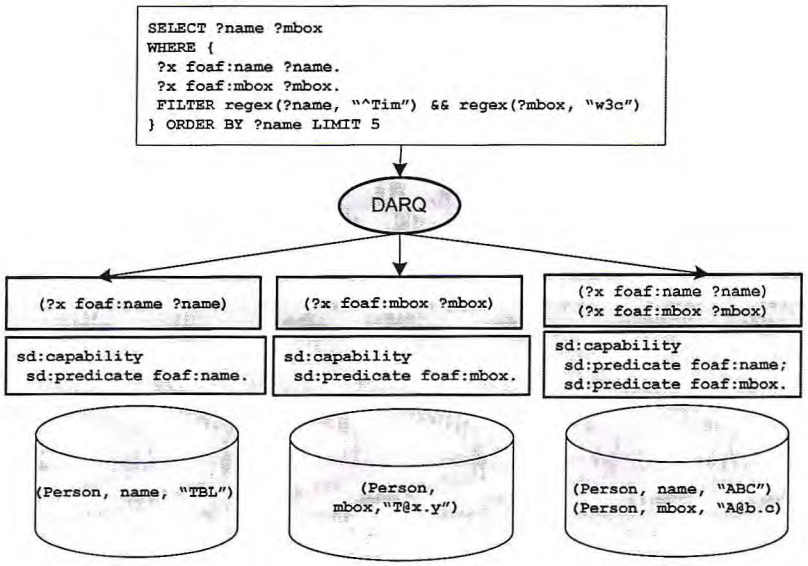


Figura 23 – Representação gráfica de reformulação de consulta no DARQ
Fonte: adaptado de QUILITZ; LESER (2008)

Para otimização do processamento de consulta, o DARQ aplica otimização lógica e física. A otimização lógica usa regras para reescrever a consulta original antes da geração do plano; e a otimização física utiliza programação dinâmica interativa, empregando informações fornecidas pelo catálogo (*service description*) para definir a ordem de execução da junção.

O DARQ implanta as seguintes operações de junção:

a) *nested-loop-join* – utiliza laços aninhados para efetuar as junções, o processamento da junção é iniciado após receber uma das entradas por completo;

b) *bind-join* – basicamente é um loop aninhado, em que os resultados intermediários de uma das fontes de dados são passados a outra consulta que participa da junção para ser usados como filtro, o objetivo é reduzir o tamanho de resultados intermediários, enviando uma consulta várias vezes seguindo o modelo tupla-a-tupla.

4.1.4 Semantic Web Integrator and Query Engine

SemWIQ (LANGEGGER, 2010), acrônimo para *Web Semântica Integrador and Query Engine*, é outro sistema de integração de dados em que as consultas são expressas em SPARQL e consideram OWL como o vocabulário para os dados RDF. Também foi implementado usando o *framework* Jena/ARQ baseado em uma arquitetura de *mediadores-*

wrappers (vide Figura 24) para geração do plano de execução adotando estratégia própria de otimização.

O sistema foi desenvolvido com foco no compartilhamento de dados científico e faz parte de um projeto maior chamado *Semantic Data Access Middleware for Grids* (GSDAM), para permitir que dados científicos fornecidos por diferentes grupos de pesquisa sejam acessados de maneira transparente e de forma compartilhada (LANGEGGER; WOSS; BLOCHL, 2008).

Porém o SemWIQ pode servir a outras aplicações como sistema de integração de dados baseado em ontologias. A arquitetura foi desenvolvida considerando três princípios básicos: os dados podem ser estruturados (por exemplo, usando XML, RDF/RDFS, OWL), são geograficamente distribuídos e armazenados em formatos heterogêneos.

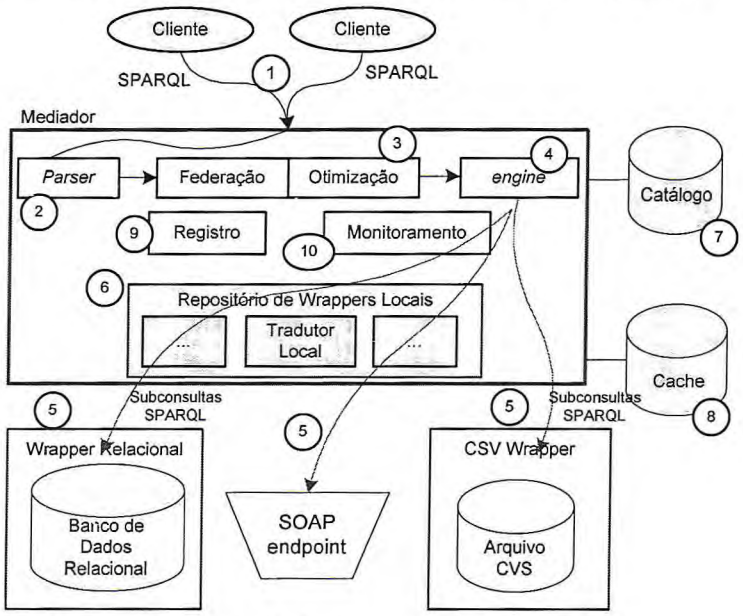


Figura 24 – Arquitetura do SemWIQ
Fonte: adaptado de LANGEGGER (2010 p.119)

Detalhes de seu funcionamento são descritos a seguir:

a) um cliente estabelece conexão com o mediador e submete uma consulta SPARQL ao esquema de mediação;

b) o tradutor, que faz parte do *framework* Jena calcula um plano de execução que é modificado pelo otimizador;

c) o otimizador analisa a consulta e procura no catálogo as fontes de dados relevantes para responder à consulta. A saída do otimizador é um plano global para a execução da consulta que é enviada ao *engine* de execução;

d) o *engine* delega a execução das subconsultas aos SPARQL *endpoints* (*wrappers*);

e) quando a fonte de dados não tem suporte nativo a consultas SPARQL é preciso um *wrapper* capaz de reescrever a consulta original, escrita em SPARQL, para o formato específico da fonte de dados consultada (disponível em formato CSV ou bancos de dados relacionais);

f) o mediador provê um *wrapper* local quando a fonte de dados é acessada por um serviço Web para envio dos dados – é importante lembrar que o SPARQL suporta o protocolo SOAP para entrega dos dados (vide seção 2.7.3);

g) o catálogo armazena descrições e estatísticas sobre as bases de dados registradas;

h) um cache local para dados RDF será usado para armazenar triplas comumente acessadas nos resultados das consultas visando melhorar o tempo de processamento das operações de junção;

i) o registro de uma fonte de dados é realizado enviando uma requisição HTTP POST para o mediador junto com um documento RDF anexado. Ele especifica o URI do SPARQL *endpoint* e possui metadados que podem conter informações da origem dos dados;

j) finalmente, o componente de *monitoramento* atualiza as estatísticas sobre as fontes de dados registradas que enviam periodicamente consultas SPARQL aos *endpoints* a fim de gerar estatísticas. Para tanto, foi criado um serviço de *pooling* para a geração de estatísticas e histogramas para descrever cada fonte de dados. Dessa forma, a descrição de cada fonte de dados é atualizada quase em tempo real. Porém, com grandes fontes de dados, o processo de *polling* pode sobrecarregar o mediador e os recursos de rede.

Esse componente de *monitoramento* é um projeto à parte chamado de *RDF-Stats* (LANGEGGER; WOSS, 2009), disponível como componente separado para o processador de consultas distribuídas que usa uma sintaxe estendida SPARQL e permite a construção de consultas de agregação (*count, min, max*). Langegger et al. (2008) afirmam que o componente de monitoramento recupera grande quantidade de dados a partir de fontes de dados e deve ser instalado próximo (ou, idealmente, na mesma máquina em que reside o *endpoint*) das fontes de dados que está monitorando. SemWIQ implanta diversas estratégias de otimização de

consultas que, em suma, consistem principalmente em regras para mover para baixo filtros ou operadores unários no plano de consulta, juntamente com a reordenação de junção baseada na aplicação de uma algoritmo de programação dinâmica interativo.

4.2 Discussão

Esta seção apresenta uma análise comparativa considerando as etapas comuns no processamento de consultas em sistemas de integração de dados.

4.2.1 Reformulação de consulta

Conforme já mencionado, segue-se a abordagem GAV para definição dos mapeamentos (denominados de mapeamentos de mediação). O algoritmo de reformulação de consulta proposto neste trabalho não faz a expansão da consulta, ou seja, não usa os axiomas da *ontologia de domínio* para a expansão da consulta durante o processo de reformulação, como foi feito em Poggi et al. (2008) e Calvanese et al. (2008). Em vez disso, o algoritmo proposto (vide seção 7.3) usa apenas os mapeamentos de mediação para o *unfolding* da consulta. Isso acontece porque os mapeamentos definem completamente o esquema de mediação, isto é, referem-se à ontologia de domínio com uma ou mais ontologias de aplicação, incluindo (*subsuming*) a informação inferida pelos axiomas da ontologia de domínio. Dessa forma, todo o conhecimento que é relevante para responder à consulta será descoberto nesta etapa de *unfolding*.

Ao utilizar o DARQ, não é preciso conhecer detalhes das fontes de dados, bastando especificar uma consulta SPARQL padrão – DARQ analisa a consulta, determina quais fontes de dados devem ser consultadas e gera um plano que representa uma consulta federada para múltiplos SPARQL *endpoints*. Desse ponto de vista, é semelhante ao algoritmo de reformulação de consulta proposto neste trabalho.

A reformulação de consulta no DARQ, diferentemente da solução proposta neste trabalho, que utiliza mapeamentos, é baseada na *descrição de serviços* que possui todas as propriedades RDF que podem ser fornecidas pelas fontes de dados, como comentado

anteriormente. Por outro lado, SemWIQ usa um serviço de *pooling* que envia consulta aos SPARQL *endpoints* para recuperar todos os nomes de classe e propriedade. Dessa forma, a descrição de cada serviço é atualizada praticamente em tempo real, contornando a limitação do DARQ.

Uma limitação desses dois enfoques é que os resultados podem ser incompletos. Não se pode simplesmente juntar os resultados das subconsultas, pois o DARQ e o SemWIQ não tratam explicitamente as ligações interontologia. Propõe-se neste trabalho um algoritmo de reformulação de consulta que usa ligações (*URI-link* e *same-as*) para extrair propriedades localizadas em outras fontes de dados expandindo assim o escopo do espaço de busca de forma a obter melhores resultados no momento em que dados de várias fontes são acessados, relacionados e combinados.

As ligações (*URI-links*) conectam recursos localizados em diferentes fontes de dados no padrão de *Linked Data* e são cruciais para os *Liked Data* visto que eles permitem juntar as fontes de dados dispersas em um espaço global de dados (MAGALHAES et al., 2011). As ligações *same-as* são virtuais, o que possibilita complementar os dados com informações de outras fontes de dados (vide seção 5.2).

4.2.2 Coleta de estatísticas das fontes de dados

Mesmo que nenhuma descrição dos dados seja fornecida por uma fonte de dados, ainda assim é possível recuperar algumas estatísticas enviando consultas SPARQL especificamente criadas para tal propósito. Mas isso pode ser um processo laborioso, visto que inicialmente nenhum conhecimento sobre a fonte de dados como tamanho e cardinalidade de cada propriedade está disponível. Então o primeiro passo é explorar a estrutura de dados (esquema) por meio de consulta para o tipo de dado e propriedades. Além disso, funções de agregação (*count*, *max*, *min*, *avg*) são necessárias para recuperar algumas estatísticas simples – essas funções serão aceitas somente a partir da SPARQL 1.1. Mesmo tendo tais funções disponíveis, o rastreamento de dados é computacionalmente caro. Como alternativa para análise dos dados, é possível extrair dados estatísticos dos resultados recuperados de uma consulta. Esta abordagem tem a sobrecarga de processamento extra, mas a vantagem de ser não-intrusiva.

Isso em virtude de os SPARQL *endpoints* poderem ser acessados por diferentes aplicações e usuários de forma aberta e operar em um ambiente distribuído que pode modificar-se constantemente. Portanto, técnicas tradicionais e algoritmos para processamento distribuído de consultas precisam ser revistos. Os projetos apresentados, com estratégias puramente estáticas, são insuficientes para modelar tais variações.

Devido a esse fato, o tráfego atual nos SPARQL *endpoints* tem sido intenso e possui requisitos operacionais variados. É importante destacar que a *Internet* é capaz de fornecer um serviço de melhor esforço (*best-effort*) (HUNT, 2002), ou seja, tentará, de todas as formas, entregar os dados que lhe foram confiados no menor tempo possível e sem erros, mas não será dada nenhuma garantia de que isso realmente ocorrerá. O problema está nos momentos de congestionamento, quando o SPARQL *endpoint* pode apresentar atraso na entrega dos dados e flutuações de desempenho no processamento das consultas.

Essa variação requer que o acesso aos SPARQL *endpoints* considere aspectos dinâmicos. Para tanto, definiu-se uma estratégia de execução adaptativa. Outro ponto-chave é que a quantidade de dados no padrão de *Linked Data* tem crescido sobremaneira nos últimos anos, e não se veem sinais de que essa tendência vá reverter em um futuro próximo. O crescimento é decorrência do aparecimento de dados abertos disponíveis livremente que podem ser consumidos por qualquer aplicação via SPARQL *endpoint*.

4.2.3 Estratégias de junção

Diferentemente dos trabalhos apresentados, são propostos neste trabalho algoritmos de junção que consideram características do ambiente Web que se adaptem ao novo cenário de execução da consulta, para reagir a latências não previstas. Utilização de algoritmos que possuem mais de uma fase de execução e não apresentam suporte para a técnica de paralelismo e/ou *pipelining*. Por exemplo, o algoritmo *nested-loop-join* usado no DARQ e SemWIQ possui duas fases de execução (construção e comparação). A fase de comparação só pode ser realizada após a execução da fase de construção por completo (ou seja, para todas as tuplas resultantes de uma das operações de junção). O desempenho deste algoritmo é afetado caso ocorra atraso na entrega dos dados, fazendo com que a primeira fase forneça o resultado tardiamente para a fase de comparação. Logo, a ideia é revisitar alguns dos algoritmos de junção propostos pela comunidade de banco de dados distribuído/paralelo.

Para tanto se utilizam os seguintes algoritmos de junção:

a) *pipelining hash join,* criado inicialmente para banco de dados paralelos, este algoritmo apresenta duas características relevantes no acesso aos *Linked Data* no ambiente Web: 1) produzir resultados incrementalmente à medida que ficam disponíveis, sem esperar que uma das entradas seja lida por completo; 2) permitir uma execução contínua do algoritmo de junção.

b) *set-bind-join,* adaptação do algoritmo *bind-join*, este algoritmo procura utilizar os benefícios do *semi-join* para reduzir o volume de dados transferidos de um SPARQL *endpoint* para o mediador.

Pode-se perceber que essa abordagem é bem diferente do projeto DARQ que adota os algoritmos de junção *nested-loop-join,* em que a geração dos primeiros resultados só ocorre após receber uma das entradas por completo. Já o *bind-join* pode aumentar bastante a troca de mensagens do SPARQL *endpoint* para o mediador, visto que, para cada *bind*, uma nova consulta e mensagem de resultado tem de ser processada, seguindo um modelo tupla-a-tupla.

4.2.4 Execução e consolidação dos resultados

As pesquisas mencionadas nas seções 4.1.3 e 4.1.4 focam no modelo de custo clássico para integração de dados a fim de melhor estimar o custo de execução das subconsultas. O esforço é para captar previamente informações de fontes de dados, com a intenção de construir um plano de execução ideal. Uma possível solução é a noção de capacidades (*capabilities*) proposta no DARQ (QUILITZ; LESER, 2008). Com base nessas capacidades, o mediador é capaz de determinar as fontes de dados relevantes para responder à consulta do usuário e otimizar as operações de junções com base na seletividade. Enquanto o projeto DARQ usa informações de seletividade fornecidas previamente, Langegger (2010) apresentou uma abordagem que se baseia em estatísticas RDF mais representativas, incluindo histogramas. As estatísticas são geradas automaticamente e podem ser solicitadas por meio de uma extensão para o protocolo SPARQL, porém o inconveniente dessa solução é que para cada SPARQL *endpoint* se deve associar um coletor de estatísticas.

De fato, tendo informações estatísticas de qualidade, a definição de um plano de execução otimizado torna-se muito mais precisa. Todavia, o mesmo o plano ideal enfrenta o problema de se tornar ineficiente em tempo de execução devido: a) à ocorrência de eventos imprevisíveis durante a execução das subconsultas; b) às variações no tempo de resposta dos SPARQL *endpoints*; e c) a falta de previsão do volume de dados retornado por uma consulta pode comprometer o desempenho do processamento das subconsultas.

Estratégias puramente estáticas não são capazes de modelar tais variações. Já uma estratégia adaptativa pode capturar eventos imprevisíveis que podem ocorrer durante a execução de uma consulta recorrendo aos algoritmos que sejam mais apropriados a cada instante. Nesse sentido, houve outro direcionamento nesta pesquisa em relação a trabalhos relacionados (QUILITZ; LESER, 2008; LANGEGGER, 2010). A solução adotada é o desenvolvimento de uma estratégia simples que muda em tempo de execução o algoritmo de junção sem descartar os dados já recebidos, em resposta à ocorrência de eventos durante a execução das consultas, como, quantidade de dados intermediários ou atraso no tempo de resposta dos SPARQL *endpoints*. Explora-se diversos níveis de paralelismo e *pipelining* e também se beneficia dos modelos de entrega de dados *push* e *pull,* e um modelo de notificação de eventos.

Vale ressaltar que o problema de processamento adaptativo de consultas não é novo, e para contorná-lo foram desenvolvidos inúmeros métodos, que geralmente começam com um plano de execução e tentam mudar de plano com melhor conhecimento das condições de tempo de execução e dos dados recebidos. Características de tais métodos são dadas em Hellerstein et al (2000), que afirmam que uma otimização de consultas é adaptativa se: a) receber informações de seu ambiente; b) usar as informações para determinar o próprio comportamento; e c) esse processo se repetir ao longo do tempo gerando *feedback* entre o mediador e o comportamento das fontes de dados.

Várias técnicas podem ser utilizadas no contexto de processamento de consulta adaptativo, a melhor técnica a ser empregada depende do problema a ser resolvido e do ambiente em questão. Uma técnica é alternar entre os planos de consulta em tempo de execução (IVES; HALEVY; WELD, 2004). Outra técnica é usar os operadores especiais, como *Eddies* (AVNUR; HELLERSTEIN, 2000), que, adaptativamente, roteiam *tuplas* de entrada por meio de uma série de operadores de consulta.

Pretende-se em trabalhos futuros utilizar o modelo de consulta do *Eddy*, que define um operador de controle *n-ário*, responsável por ler tuplas de uma fonte de dados e

distribuí-las para os demais operadores do plano, que as processarão e as devolverão para serem executadas pelo próximo operador. Para tanto, será preciso definir uma política de roteamento que determina para quais operadores uma tupla pode ser encaminhada e uma política de controle de fluxo que determina para qual destes operadores a tupla deve seguir.

Para facilitar este processo, também, almeja-se, em trabalhos futuros, instanciar o *framework* (*Query Engine Execution Framework*) (AYRES; PORTO; MELO, 2003) que suporta diferentes características de execução como, por exemplo, paralelismo interoperador e intraoperador), modelo de tratamento do *streamming* e controle e estratégia de produção de resultados parciais ou completos.

4.3 Considerações do capítulo

A linguagem SPARQL é relativamente nova, sendo uma recomendação da W3C desde 2008. Trabalhos de integração de dados no padrão de *Linked Data* acessados via SPARQL *endpoints* ainda estão em estágio inicial e há espaço para melhorias. O trabalho proposto foi desenvolvido considerando a natureza dos dados no padrão de *Linked Data* tanto na etapa de reformulação de consulta quanto na etapa de execução das consultas. Os capítulos seguintes discutem em detalhes o *framework* para integração de *Linked Data* e em seguida um método de processamento distribuído de consultas SPARQL.

5 FRAMEWORK PARA INTEGRAÇÃO DE DADOS NO PADRÃO DE *LINKED DATA*

Este capítulo apresenta a arquitetura do *framework* proposto para integração de dados no padrão de *Linked Data*. É apresentado um exemplo e discutido como o *framework* ajuda a resolver algumas limitações das atuais abordagens de integração de dados baseada em ontologia.

No Capítulo 6 será apresentado um método para processamento de consultas neste *framework*.

5.1 Introdução

O *framework* proposto é baseado numa arquitetura de três níveis de esquema, que utiliza as ontologias tanto para descrição semântica das fontes de dados quanto linguagem para representar o esquema de mediação. O esquema de mediação (ontologia de domínio) fornece uma representação conceitual do domínio e tem duas funções (PINHEIRO; VIDAL, 2009): a) prover ao usuário o acesso aos dados com uma interface de consulta uniforme para facilitar a formulação da consulta; b) definir um vocabulário compartilhado com um conjunto de axiomas para descrever o conteúdo de todas as fontes de dados.

A Figura 25 ilustra os principais componentes. Cada fonte de dados, publicada na Web no padrão de *Linked Data*, é descrita por uma ontologia de aplicação (*OA*) cujo vocabulário é um fragmento fechado, ou seja, um subconjunto do vocabulário da ontologia de domínio (*OD*), o que pode simplificar consideravelmente o processo de reformulação de consultas, pois as ontologias de aplicação são baseadas nas mesmas primitivas, e nenhuma inferência é necessária. Adotou-se OWL *Lite* (BECHHOFER et al., 2004) como a linguagem para representar a ontologia de domínio e as ontologias de aplicação.

Convém enfatizar que, no caso em que a ontologia de aplicação seja um fragmento aberto, o problema de informação incompleta pode existir, e o *framework* tem que considerar os axiomas durante a execução da consulta para lidar com informação incompleta. Uma técnica baseada na expansão de consulta tem sido adotada por Calvanese et al. (2008) para tratar este problema.

Os esquemas-fonte (F_i) são acessados por meio de *wrappers*. Os *wrappers* são definidos como SPARQL *endpoints* que aceitam requisições HTTP (vide seção 2.8) para consultar as fontes de dados no padrão de *Linked Data*. Os esquemas-fonte são expressos como vocabulário, ou seja, uma coleção de classes e propriedades definidas em termos de RDFS e OWL. Os dados reais não necessariamente precisam ser armazenados em um repositório RDF, pode-se utilizar um banco de dados relacional com um *wrapper*, como o D2R Server (BIZER; CYGANIAK, 2006) que publica dados de banco de dados relacionais no padrão de *Linked Data*.

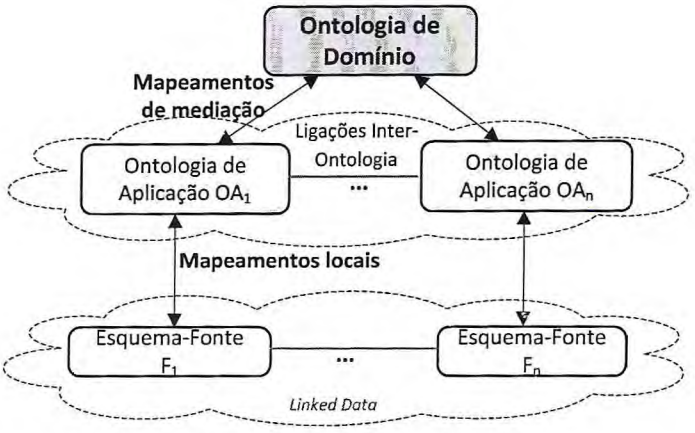


Figura 25 – Arquitetura de três níveis para integração de dados

Os mapeamentos de mediação especificam os conceitos da ontologia de domínio em termos das ontologias de aplicação, e os mapeamentos locais definem os conceitos das ontologias de aplicação em termos dos elementos de seu esquema-fonte correspondente (PINHEIRO et al., 2010).

Para o *framework* proposto, definiu-se um ambiente de mediação:

Definição 7 – um ambiente de mediação é definido como uma tripla = $(OD, (F_k, OA_k, \gamma_k)_{k=1}^{n}), \gamma)$, em que:

a) *OD* é uma *ontologia de domínio*, que representa o esquema de mediação. Assume-se que as classes e propriedades na *OD* são *C1,...,Cu* e *P1,...,Pv* respectivamente;

b) para cada *k=1,...,n*,

i. F_k é o esquema de uma fonte de dados publicado no padrão de *Linked Data*.

ii. OA_k é uma ontologia de aplicação que descreve o esquema da fonte de dados F_K. O vocabulário da OA_K é um subconjunto do vocabulário da *OD*. Adotam-se prefixos que definem espaços de nomes para distinguir a ocorrência do mesmo símbolo no vocabulário da *OD* e no vocabulário da OA_K. Assume-se que para cada classe C_i (ou propriedade P_J) no vocabulário da *OD*, tem-se a ocorrência de pelo menos uma classe C_i (ou propriedade P_J) no vocabulário da OA_K através da $OA_K{:}C_i$ (ou $OA_K{:}P_J$).

iii. γ_k é um conjunto de *mapeamentos locais* que relaciona os conceitos de uma OA_K com os elementos da F_K. Considera-se que as visões definidas pelos mapeamentos locais são exatas.

c) γ são os *mapeamentos de mediação* que seguem a abordagem GAV: definem-se classes e propriedades de uma *OD* em termos de classes e propriedades das *OAs*. Considera-se que as visões definidas pelos mapeamentos de mediação são completas. □

Na sequência, explica-se o ambiente de mediação concomitantemente com um exemplo de integração de dados no padrão de *Linked Data* provenientes de fontes de dados autônomas, heterogêneas e distribuídas geograficamente, usando um exemplo de uma loja virtual de *e-commerce* adaptado de Bizer e Schultz (2008), nas quais produtos são oferecidos por diferentes fornecedores e em que os consumidores postam comentários sobre produtos. As ontologias das Figuras 26, 28 e 29 foram ilustradas em diagramas de classe em *Unified Modeling Language* (UML).

5.2 Ontologia de domínio

A Figura 26 mostra uma representação conceitual da ontologia de domínio *Sales*. O modelo é referente a um sistema típico de *e-commerce* e contém as seguintes classes: *Product, ProductType, Producer, Offer, Review, e Person.*

Os produtos têm diversas características. Dois produtos do mesmo tipo (*ProductType*) compartilham conjunto de características idêntico. Cada produto (*Product*) possui um fabricante (*Producer*) e é ofertado para venda (*Offer*). As ofertas são válidas por um período específico e possuem preço e prazo de entrega. Os consumidores (*Person*) postam revisões (*Reviews*) sobre os produtos que consistem de título e texto opinativo. E podem ter uma avaliação com uma nota entre 1 e 10.

Usa-se o prefixo "*s:*" para se referir ao vocabulário da ontologia de domínio. Por exemplo, *s:Offer* é declarado como uma classe; *s:title* é definido como uma propriedade com domínio *s:Product* e contradomínio *string;* e *s:hasReviewer* é definida como uma propriedade de objeto com domínio *s:Review* e contradomínio *s:Person.*

A Figura 27 apresenta os axiomas de propriedade funcional inversa (*inverse functional axioms*) da ontologia de domínio *Sales*. As propriedades *s:title* e *s:name* são propriedades funcionais inversa, isto é, constituem chaves simples das classes: *Product* e *Person* respectivamente.

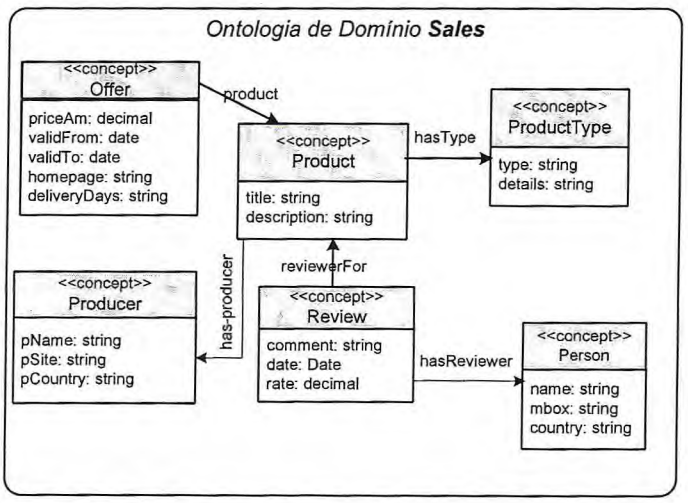


Figura 26 – Ontologia de domínio *Sales*

```
A1: (s:name rdf:type owl:InverseFunctionalProperty)
A2: (s:title rdf:type owl:InverseFunctionalProperty)
```

Figura 27 – Axiomas de Propriedade Funcional Inversa

Baseado nos axiomas de *propriedade funcional inversa* da ontologia de domínio *Sales* (Figura 27), é possível identificar automaticamente ligações *interontologia* relacionando instâncias em diferentes ontologias de aplicação. A identificação dessas ligações, denominadas de *same*-as (vide Definição 9), desempenha papel importante na geração dos mapeamentos de mediação, pois informações de uma fonte de dados podem ser complementadas por informações de fontes referenciadas por ligações. Além disso, na reformulação da consulta, essas ligações dão suporte às operações de junção, tratadas neste trabalho.

Os *InverseFunctionalProperty*, que se assemelham à noção de uma chave simples, em banco de dados, para propriedades de objeto, é uma propriedade p com domínio C e contradomínio D especifica que dadas duas instâncias $C(x1)$ e $C(x2)$, tal que $p(x1, y1)$ e $p(x2, y2)$, com $D(y1)$ e $D(y2)$, se $y1=y2$, então $x1=x2$.

Pode-se dizer que a propriedade p é uma propriedade interontologia ou uma ligação interontologia *se somente se* o domínio e contradomínio de p pertencerem a ontologias de aplicação diferentes, p é uma ligação interontologia entre V e U *sse* V seja o vocabulário do domínio e U seja o vocabulário do contradomínio de p; p é uma *propriedade interontologia* entre V e U *se somente se* p é um *InverseFunctionalProperty* em V e em U.

Define-se a seguir *URI-link* e *same-as* (VIDAL et al., 2011), que contêm informações sobre a *propriedade interontologia* e *ligação interontologia*, respectivamente.

Definição 8 *(URI-link)* – conjunto de triplas (o_i, o_j, p) em que cada o_i e o_j são ontologias de aplicação distintas e p é uma propriedade tal que $dom(p) \in o_i$ *e* $contradomínio(p) \in o_j$. □

Definição 9 *(same-as)* – conjunto de triplas (o_i, o_j, p) em que o_i e o_j são ontologias de aplicação distintas e p é uma propriedade comum em ambas as ontologias *oi* e *oj*, tal que $p \in oi, oj$. □

5.3 Esquemas-Fonte

No exemplo, os esquemas-fonte representam diretamente as fontes de dados já publicadas no padrão de *Linked Data*. Há dois esquemas-fonte para descrever os dados sobre as lojas virtuais *Amazon* e *eBay*, um terceiro esquema-fonte descreve os fabricantes dos

produtos (*Producer*) e um quarto esquema possui informações dos revisores *Foaf*[13], que é um acrônimo para *Friend of a Friend* (KRUK, 2004) escrita em OWL para descrever informações pessoais. Utiliza-se apenas uma pequena parte do vocabulário, a classes *Person* com as propriedades *name*, *homepage* e *country*.

A Figura 28 mostra uma representação conceitual dos esquemas-fonte. Usam-se os prefixos "*a:*", "*e:*", "*p:*" e "*f:*" para referir o vocabulário dos esquemas-fonte *Amazon*, *eBay*, *Producer*, e *Foaf* respectivamente.

Observe-se que a propriedade *a:reviewer* com domínio a:*Review* e contradomínio *f:Person* é uma *propriedade interontologia* porque conecta recursos de diferentes ontologias. As instâncias das propriedades *a:reviewer* são propriedades RDF que conectam uma instância de a:*Review* com uma instância de *f:Person*.

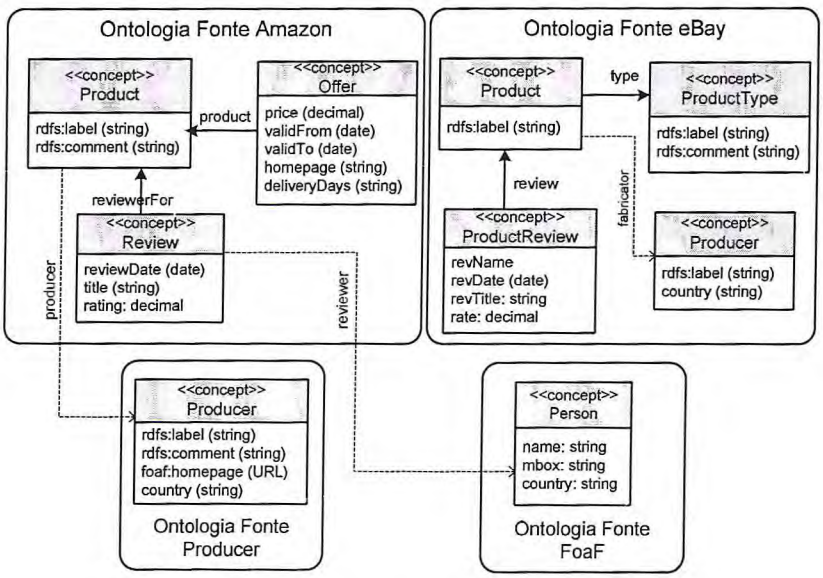


Figura 28 – Esquemas-fonte *Amazon*, *eBay*, *Producer* e *FoaF*

5.4 Ontologias de aplicação

A Figura 29 mostra uma representação conceitual da ontologia de aplicação obtida com base no resultado do *matching* entre cada esquema-fonte e a ontologia de domínio. O

[13] http://www.foaf-project.org/

vocabulário de uma ontologia de aplicação é composto de classes e propriedades que são um subconjunto da ontologia de domínio que corresponde (*matches*) ao esquema-fonte.

Para distinguir os espaços de nomes usam-se os seguintes prefixos "*am:*", "*eb:*", "*pr:*" e "*fp:*" para se referir aos vocabulários das ontologias de aplicação *Amazon*, *eBay*, *Producer* e *Foaf* respectivamente.

Observe-se que a propriedade *am:has-producer* com domínio a:*Review* e contradomínio *f:Producer* é uma *propriedade interontologia* porque conecta recursos de diferentes ontologias. As instâncias das propriedades *a:reviewer URI-links* (vide Definição 8) RDF que conectam uma instância de a:*Review* com uma instância de *f:Producer*.

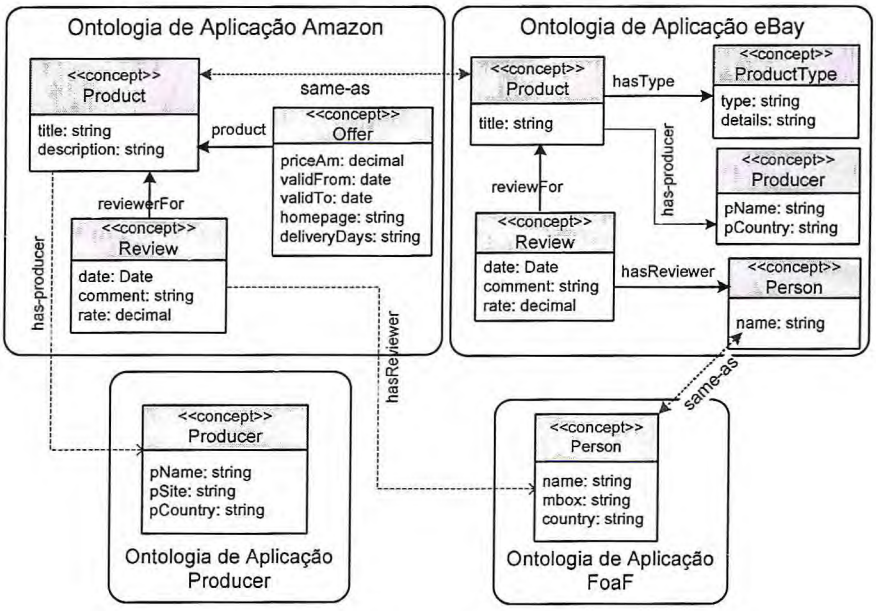


Figura 29 – Ontologias de aplicação

O axioma *A1* especifica uma ligação *interontologia* (propriedade virtual) *same-as*, que relaciona instâncias de *eb:Person* e *fp:Person*, definido da seguinte forma:

$$(A1)\ same\text{-}as(X,Y) \Leftarrow eb{:}Person(X),\ eb{:}name(X,n),\ fp{:}Person(Y),\ fp{:}name(Y,n)$$

De maneira semelhante, o axioma *A2* especifica uma propriedade virtual *same-as*, que relaciona instâncias de *am:Product* e *eb:Product*, definidos como segue:

(A2) same-as(X,Y) ⇐ am:Product(X), am:title(X,t),
eb:Product(Y), eb:title(Y,t)

Ademais, as ontologias de aplicação são comparáveis porque se baseiam nas mesmas primitivas, o que é muito importante para a solução proposta, pois facilita a reformulação de consultas.

Portanto, as ontologias de aplicação ajudam a dividir o problema de reformulação de consultas e fornecem uma notação apropriada para segmentar a definição dos mapeamentos em duas etapas: a definição dos mapeamentos de mediação e a definição dos mapeamentos locais, apresentadas na próxima seção.

Sacramento et al. (2010) apresentaram um modelo para a especificação de *matching* (i.e. operação que permite a comparação para identificação de similaridade) do vocabulário de ontologias. O objetivo é descrever as correspondências entre cada esquema das fontes de dados e o esquema da ontologia de domínio e utilizar essas correspondências para geração automática das ontologias de aplicação e os respectivos mapeamentos de mediação e locais, que serão apresentados na próxima seção.

5.5 Mapeamentos

Assim como dito no Capítulo 3, os mapeamentos entre ontologias podem ser homogêneos ou heterogêneos e com relação à direção, um mapeamento pode ser classificado como unidirecional ou bidirecional.

Como, neste trabalho, as ontologias de aplicação são fragmentos de uma ontologia de domínio, os mapeamentos de mediação são homogêneos, os mapeamentos locais são heterogêneos, sendo todos eles unidirecionais.

5.5.1 Mapeamentos de mediação

Os mapeamentos de mediação (γ) definem o esquema de mediação, isto é, relacionam à ontologia de domínio com uma ou mais ontologias de aplicação. Eles são criados para garantir que uma consulta SPARQL seja reescrita corretamente.

Além disso, os mapeamentos de mediação permitem a definição de uma classe (propriedade) de uma ontologia de domínio por meio de um axioma único, composto pela união de classes (propriedades) de uma ou mais ontologias de aplicação. Tais mapeamentos expressam mapeamentos homogêneos entre ontologias, pois podem ser utilizados para o *unfolding* de uma consulta submetida sobre a ontologia de domínio em um ou mais subconsultas sobre as ontologias de aplicação.

A Figura 30 mostra os mapeamentos de mediação que relacionam as ontologias de aplicação com a ontologia de domínio usando a abordagem GAV. Observe o uso de ";" para denotar regras disjuntivas.

Classes:

```
s:Product(p) ⇐ am:Product(p); eb:Product(p)
s:Review(r) ⇐ am:Review(r); eb:Review(r)
s:Offer(o) ⇐ am:Offer(o)
s:Person(ps) ⇐ eb:Person(ps); fp:Person(ps)
s:ProductType(pt) ⇐ eb:ProductType(pt)
s:Producer(pdr) ⇐ eb:Producer(pdr); pr:Producer(pdr)
```

Propriedades:

```
s:title(p, t) ⇐ am:title (p, t); eb:title(p, t)
s:description(p, d) ⇐ am:description(p, d)
s:has-producer(p, pdr) ⇐ am:has-producer(p, pdr); eb:has-producer(p, pdr)
s:pName(pdr, pn) ⇐ pr:pName(pdr, pn)
s:pSite(pdr, s) ⇐ pr:pSite(pdr, s)
s:pCountry(pdr, cy) ⇐ eb:pCountry(pdr, cy); pr:pCountry(pdr, cy)
s:comment(r, c) ⇐ am:comment(r, c); eb:comment(r, c)
s:date(pdr, dt) ⇐ am:date(pdr, dt); eb:date(pdr, dt)
s:rate(r, rt) ⇐ am:rate(r, rt); eb:rate(r, rt)
s:hasReviewer(r, rvr) ⇐ am:hasReviewer(r, rvr); am:Review(r)
s:reviewFor(r, p) ⇐ am:reviewFor(r, p); am:Review(r)
s:name(ps, n) ⇐ eb:name(ps, n); fp:name(ps, n)
s:mbox(ps, m) ⇐ fp:mbox(ps, m)
s:country(ps, ct) ⇐ fp:country(ps, ct)
s:priceAm(o, pa) ⇐ am:priceAm(o, pa)
s:validFrom(o, vf) ⇐ am:validFrom(o, vf)
s:validTo(o, vt) ⇐ am:validTo(o, vt)
s:homepage(o, h) ⇐ am:homepage(o, h)
s:deliveryDays(o, dd) ⇐ am:deliveryDays(o, dd)
s:product(o, pd) ) ⇐ am:product(o, pd)
s:hasType(p, pt) ⇐ eb:hasType(p, pt)
s:type(pt, ty) ⇐ eb:type(pt, ty)
s:details(pt, ds) ⇐ eb:details(pt, ds)
```

Figura 30 – Mapeamentos de mediação

5.5.2 Mapeamentos locais

Especificam a correspondência entre cada ontologia de aplicação com o correlacionado esquema-fonte. É importante destacar que esses mapeamentos são heterogêneos e representam relacionamentos 1:1 e permitem traduzir uma consulta sobre uma ontologia de aplicação em termo do esquema-fonte correspondente, esta operação é chamada de *Data Exchange*.

Libkin e Sirangelo (2011) definiram formalmente o problema de *data exchange*. As regras de mapeamento são semelhantes aos mapeamentos de mediação, exceto para os prefixos de espaço de nomes que devem referir-se ao vocabulário da ontologia de aplicação e o nome das propriedades que são heterogêneos.

A Figura 31 exibe os mapeamentos para os esquemas-fonte e as ontologias de aplicação ilustradas nas Figuras 28 e 29, respectivamente.

(γ_1): EF Amazon – OA Amazon	(γ_2): EF eBay – OA Ebay
am:Product(p) ⇐ a:Product(p) am:Review(r) ⇐ a:Review(r); am:Offer(o) ⇐ a:Offer(o) am:hasReviewer(r, rvr)⇐ a:reviewer(r, rvr) am:reviewFor(r, p) ⇐ a:reviewFor(r, p) am:product(o, p) ⇐ a:product(o, p) am:has-producer(pdr) ⇐ a:producer(pdr) am:title(p, t) ⇐ rdfs:label(p, t) am:description(p, d)⇐ rdfs:comment(p, d) am:date(r, dt) ⇐ a:reviewDate(r, dt) am:comment(r, cm) ⇐ a:title(r, cm) am:rate(r, rt) ⇐ a:rating(r, rt) am:priceAm(o, pa) ⇐ a:price(o, pa) am:validFrom(o, vf) ⇐ a:validFrom(o, vf) am:validTo(o, vt) ⇐ a:validTo(o, vt) am:homepage(o, h) ⇐ a:homepage(o, h) am:deliveryDays(o, dd) ⇐ a:deliveryDays(o, dd)	eb:Product(p) ⇐ e:Product(p) eb:Review(r) ⇐ e:ProductReview(r) eb:ProductType(pt) ⇐ e:ProductType(pt) eb:Producer(pdr) ⇐ e:Producer(pdr) eb:hasReviewer(r, rvr)⇐ e:revName(r, n); e:ProductReview(r); eb:Person(*fperson*(n)) ⇐ e:revName(r, n); e:ProductReview(r); eb:name (*fperson* (n),n) ⇐ e:revName(r,n),e:ProductReview(r) eb:reviewFor(r, p) ⇐ e:review(r, p) eb:title(p, t) ⇐ rdfs:label(p, t) eb:has-producer(pdr) ⇐ e:fabricator(pdr) eb:date(r, dt) ⇐ e:revDate(r, dt) eb:comment(r, cm) ⇐ e:revTitle(r, cm) eb:rate(r, rt) ⇐ e:rate(r, rt) eb:hasType(p, pt) ⇐ e:hasType(p, pt) eb:type(pt, ty) ⇐ rdfs:label(pt, ty) eb:details(pt, ds) ⇐ rdfs:comment(pt, ds)
(γ_3): EF FoaF – OA Person	(γ_4): EF Producer – OA Producer
fp:Person(ps) ⇐ f:Person(ps) fp:name(ps, n) ⇐ f:name(ps, n) fp:mbox(ps, m) ⇐ f:mbox(ps, m) fp:country(ps, ct) ⇐ f:country(ps, ct)	pr:Producer(pdr) ⇐ p:Producer(pdr) pr:pName(pdr, pn) ⇐ rdfs:label(pdr, pn) pr:pSite(pdr, s) ⇐ foaf:homepage (pdr,s) pr:pCountry(pdr,cy) ⇐ p:country(pdr,cy)

Figura 31 – Mapeamento locais

Nota-se que a ontologia de aplicação *eBay* é obtida com base tanto nos mapeamentos da Figura 31 quanto nos axiomas da ontologia de domínio *Sales* da Figura 26. Observe também que o esquema fonte *eBay* não tem a classes *Person* (vide Figura 28). Assim, os axiomas da ontologia de domínio *Sales* foram usados para modelar as limitações da nova classe *Person* na ontologia de aplicação *eBay*. Sacramento et al. (2010) apresenta um formalismo destas regras de mapeamento.

5.6 Considerações do capítulo

O objetivo principal deste capítulo foi apresentar um *framework* para integração de dados no padrão de *Linked Data,* incluindo os benefícios obtidos com uso de ontologias para realizar a integração semântica, o enfoque híbrido e a arquitetura de três níveis se mostraram mais adequado aos nossos objetivos, uma vez que melhor reflete a realidade de contarmos com fontes de dados heterogêneas e simplificar a integração de dados no padrão de *Linked Data*, uma vez que as ontologias de aplicação são um fragmento homogêneo da ontologia de domínio, e nenhuma inferência é necessária. Para lidar com informação incompleta, ligações *same-as* e URI *links* são considerados durante a reformulação da consulta, o que gera resultados mais significativos e completos para o usuário.

As ontologias de aplicação ajudam a dividir o problema de processamento de consultas, no contexto de integração de dados, em dois subproblemas (menos desafiadores): a) responder a consultas no âmbito de fragmentos homogêneas, isto é, como reformular uma consulta sobre a ontologia de domínio em um ou mais subconsultas expressas em termos das ontologias de aplicação (fragmentos homogêneos), b) como responder a consultas no contexto de *data-exchange,* ou seja, como transformar subconsultas sobre uma ontologia de aplicação para uma consulta sobre o esquema-fonte.

6 MÉTODO DE PROCESSAMENTO DISTRIBUÍDO DE CONSULTAS SPARQL

Este capítulo apresenta um método para o processamento de consultas SPARQL baseado no *framework* descrito no Capítulo 5. Inicialmente, serão apresentados alguns conceitos da área de processamento distribuído de consultas. Depois, enuncia-se uma visão geral do método proposto, seguido por cada etapa. A etapa de tradução é apresentada na seção 6.3. A etapa de reformulação da consulta, que envolve também a eliminação de fontes com dados irrelevantes para o resultado da consulta, é vista na seção 6.4, enquanto a seção 6.5 analisa brevemente a etapa de execução das subconsultas e a composição do resultado final. As etapas de reformulação e execução serão individualizadas no Capítulo 7 e 8 respectivamente.

6.1 Introdução

O processamento de consultas, em qualquer sistema de integração, tem papel de destaque, pois subconsultas destinadas a fontes de dados autônomos possibilitam execução de forma concorrente/paralela e, consequentemente, podem melhorar o tempo de resposta ao usuário. Porém, o processamento eficiente de uma consulta depende de vários fatores combinados (KOSSMANN, 2000). Devem-se considerar – além do tempo de processamento e das operações de E/S – os seguintes fatores: o custo da transmissão dos dados e a capacidade de processamento de cada SPARQL *endpoint*. Em relação ao custo de transmissão dos dados, deve-se evitar transmitir grande quantidade de dados pela rede devido ao custo de transmissão e sobrecarga de processamento no mediador.

Além disso, deve-se garantir a transparência da localização física dos dados aos usuários e às aplicações finais, assim como assegurar a precisão na tradução de uma consulta expressa em termos de uma ontologia de domínio em subconsultas destinadas aos diversos SPARQL *endpoints* e à execução do plano gerado. Dessa forma, os usuários poderão formular consultas sem nenhuma preocupação com a reconstrução dos dados, deixando essa árdua tarefa a cargo do mediador, que é responsável pela manutenção de metadados sobre as fontes de dados e a execução das consultas.

As diferentes etapas do processamento de uma consulta em um ambiente distribuído, como os sistemas baseados em mediadores, podem ser continuamente evoluídas,

sendo os modelos relacionais (OZSU; VALDURIEZ, 1999) e XML (TEIXEIRA, 2009), já com muitos anos de utilização, os mais conhecidos. No entanto, com o aumento do volume de dados no padrão de *Linked Data*, tem crescido o interesse pela utilização de técnicas de banco de dados distribuídas para o processamento distribuído de consultas SPARQL (HARTIG; BIZER; FREYTAG, 2009; LANGEGGER; WOSS; BLOCHL, 2008; QUILITZ; LESER, 2008).

6.2 Visão geral do método proposto

O método proposto está baseado na metodologia para o processamento de consultas SQL no modelo relacional (OZSU; VALDURIEZ, 1999) e consiste em três etapas, resumidas como segue: tradução, reformulação da consulta, execução/consolidação dos resultados.

É importante observar que a autonomia dos SPARQL *endpoints*, que são públicos e acessados livremente por qualquer usuário ou aplicação na Web com uma simples requisição HTTP GET ou POST, e a facilidade de atualização de dados publicados no padrão de *Linked Data* torna difícil obter e manter informações estatísticas de qualidade sobre os dados disponibilizados. Essas características fazem de uma etapa de otimização baseada em funções de custo inadequada para o domínio deste trabalho.

Resumidamente, o mediador realizará o processamento de uma consulta SPARQL sobre a ontologia de domínio para gerar subconsultas federadas destinadas a múltiplos SPARQL *endpoints* autônomos. Cada *endpoint* irá gerenciar a execução de sua subconsulta. O mediador fará a integração e a consolidação de todos os resultados dos SPARQL *endpoint* e enviará o resultado final ao usuário (PINHEIRO et al., 2010), proporcionando, dessa forma, acesso transparente às fontes de dados como se fosse um único grafo RDF.

SPARQL é uma linguagem com ampla gama de formatos de construtores e de consulta. As consultas podem ser formuladas usando os conceitos fornecidos pela ontologia de domínio, para tanto, utilizando diretamente a linguagem SPARQL ou através de uma interface gráfica em que o usuário seleciona os conceitos e propriedades da ontologia de domínio. Na abordagem proposta, considera-se um conjunto restrito de consultas SPARQL, e essas consultas devem incluir, como primeira restrição na cláusula *WHERE,* um conceito primário identificado pelo predicado *rdf:type*. A definição é apresentada a seguir:

Definição 10 (Consulta *Q*) – é uma consulta expressa sobre a ontologia de domínio (*OD*), tendo a seguinte forma: Q ≡ ***SELECT*** *varlist* ***WHERE*** (*C* ⊓ *gp*), onde *C* representa um conceito primário, identificado pelo predicado *rdf:type*; *gp* representa um padrão de grafo que adiciona restrições sobre *C*, e *varlist=v1;...;vn,* para $n \geq 1$, é uma lista ordenada de variáveis e $varlist \subseteq var(gp)$; a função *var(gp)* retorna o conjunto de variáveis que ocorrem em *gp*. □

Por exemplo, considere-se uma consulta SPARQL ***Q*** sobre a ontologia de domínio *Sales* (Figura 26), a qual recupera as revisões [títulos (*?t*), comentários (*?cm*) e notas (*?rt*)] dos produtos fabricados nos Estados Unidos e avaliados por consumidores da França, cuja *nota* é maior do que oito (*?rt* > *8*). A Figura 32 reflete essa consulta.

```
PREFIX s:<http://sales/>
PREFIX rdf:<...>

SELECT ?t ?cm ?rt   varlist
WHERE {
 ?r rdf:type s:Review .   C
 ?r s:comment ?cm .                          gp
 ?r s:rate ?rt . FILTER (?rt > 8 )
 ?r s:hasReview ?rvr .
 ?rvr s:country ?ct . FILTER(?ct, 'FR')
 ?r s:reviewFor ?p .
 ?p s:title ?t .
 ?p s:has-producer ?pc .
 ?pc s:pCountry ?cy FILTER(?cy, 'USA')
}
```

Figura 32 – Consulta SPARQL ***Q*** sobre a ontologia de domínio *Sales*

Nas próximas seções, apresentam-se cada etapa do método juntamente com um exemplo para ilustrá-las.

6.3 Tradução

Primeiramente, é feita uma validação sintática e semântica da consulta ***Q***. A validação sintática consiste em verificar se a consulta original, expressa em SPARQL, está em conformidade com a Definição 10. São exemplos de erros de sintaxe: palavras-chave da linguagem escritas ou posicionadas de forma errada na consulta. Já a validação semântica consiste na verificação de que a consulta está de acordo com o esquema da ontologia de

domínio utilizada. Nomes de classes ou propriedades inexistentes no esquema da ontologia de domínio são exemplos de erros semânticos.

Consultas invalidadas sintaticamente são descartadas pelo mediador, pois seu processamento é impossível. Neste caso, o usuário receberá uma mensagem de erro informando o problema detectado para que possa corrigir a consulta submetida.

Em seguida, a consulta Q é transformada numa árvore que representa a estrutura da consulta sobre a ontologia de domínio, conforme a Figura 33.

Cada nó da árvore é marcado com um tipo de dado ou variável de objeto. Cada aresta é rotulada com uma propriedade e cada nó tem uma anotação com o respectivo tipo. O nó-raiz da árvore corresponde ao conceito primário C (vide Definição 10).

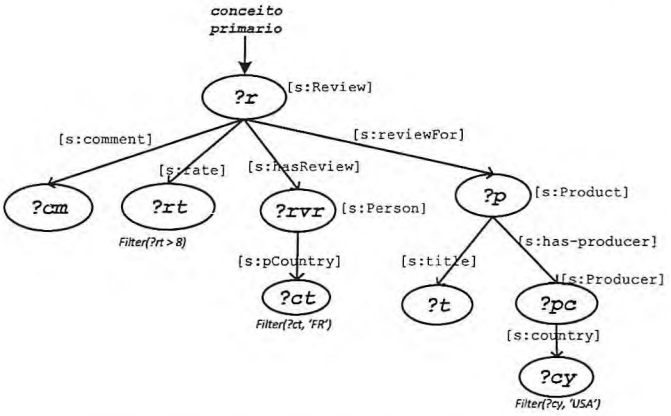


Figura 33 – Árvore da consulta Q apresentada na Figura 32

Esta representação inicial em árvore da consulta em termos da ontologia de domínio é pré-requisito para a próxima etapa de reformulação de consulta.

6.4 Reformulação de consulta

A árvore gerada no passo anterior é reformulada em uma nova árvore, que é uma combinação de subconsultas sobre as fontes de dados relevantes. Cada subconsulta objetiva extrair dados de uma única fonte de dados. Todos os conceitos (propriedades) que sejam relevantes para responder à consulta são descobertos nesta etapa.

A reformulação da consulta é dividida em duas subetapas:

a) a consulta ***Q*** é reformulada, com base nos mapeamentos de mediação, em um plano de execução de consultas sobre as ontologias de aplicação. Cada subconsulta ***Qi*** visa a extrair dados de uma única ontologia de aplicação;
b) cada subconsulta ***Qi*** é, então, traduzida em termos de uma consulta sobre as fontes de dados com base nos mapeamentos locais.

Os mapeamentos são fundamentais nesta etapa, pois eliminam previamente fontes de dados irrelevantes para a consulta, possibilitando desempenho melhorado devido à diminuição do volume de dados consultados.

Inicialmente, a árvore de consulta ***Q*** da Figura 33 é reformulada gerando um plano de execução da consulta, representado por uma árvore formada basicamente por operações de união e junção. Sendo que, o operador *Union* é definido na álgebra SPARQL (PÉREZ; ARENAS; GUTIERREZ, 2009) e o operador *Sequence* definido no Jena/ARQ (CARROLL et al, 2004), especificamente, para executar operações de junções sobre múltiplas fontes de dados. O operador *service* foi definido recentemente (BUIL-ARANDA; ARENAS; CORCHO, 2011) e indica cada SPARQL *endpoint* que será consultado. Todos estes operadores estão presentes na mais recente documentação do SPARQL PRUD'HOMMEAUX; BUIL-ARANDA, 2011).

A Figura 34 apresenta a árvore de consulta reformulada, detalhada a seguir do nó-raiz aos nós-folhas:

a) as operações de integração união (*Union*) e junção múltipla (*Sequence*) para garantir a reconstrução das instâncias da ontologia de domínio a partir de instâncias obtidas das fontes de dados;

b) o nó (*service*) representa a URI dos SPARQL *endpoints* que executam as subconsultas correspondentes; e

c) os nós-folha representam uma subconsulta sobre uma única ontologia de aplicação obtida pelas operações de *unfolding* dos mapeamentos de mediação.

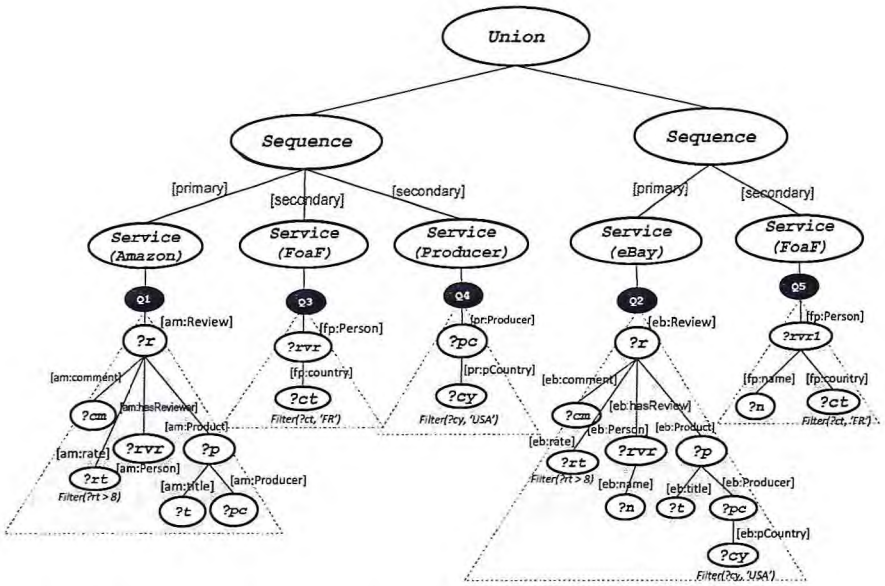


Figura 34 – Plano de execução da consulta *Q*

Observando a árvore de consulta da Figura 34, para cada nó *Service* tem-se uma subárvore que representa uma subconsulta sobre uma ontologia de aplicação. O plano de execução é composto por cinco subconsultas: *Q1, Q2, Q3, Q4* e *Q5*. Cada subárvore é reescrita numa consulta SPARQL, ilustrada na Figura 35, que visa a extrair dados de uma única ontologia de aplicação.

Por exemplo, observando-se a consulta *Q2,* vê-se que a cláusula *SELECT* é formada pelas variáveis *?t ?cm ?rt* que contribuem para resposta da consulta e pela variável de junção *?n*, que neste caso será utilizada com critério de junção como resultado da consulta *Q5*, e a cláusula *WHERE* reflete cada subárvore por meio de um padrão de grafo (vide **Definição 5**, Capítulo 2), que é formado por padrões de triplas (vide **Definição 4**, Capítulo 2).

E cada padrão de tripla é construído da seguinte maneira: o nó-pai (uma variável) representa o sujeito; a aresta representa um predicado; e o nó-filho (variável) representa o objeto.

Q1
```
SELECT ?t ?cm ?rt ?rvr ?pdr
WHERE {
 ?r rdf:type am:Review .
 ?r am:comment ?cm .
 ?r am:rate ?rt .
 FILTER (?rt > 8 )
 ?r am:hasReviewer ?rvr .
 ?r am:reviewFor ?p .
 ?p am:title ?t .
 ?p am:has-producer ? pdr.
}
```

Q2
```
SELECT ?t ?cm ?rt ?n
WHERE {
 ?r rdf:type eb:Review .
 ?r eb:comment ?cm .
 ?r eb:rate ?rt .
 FILTER (?rt > 8 )
 ?r eb:hasReviewer ?rvr .
 ?rvr eb:name ?n .
 ?r eb:reviewFor ?p .
 ?p eb:title ?t .
 ?p eb:has-producer ?pdr .
 ?pdr eb:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```

Q3
```
SELECT ?rvr
WHERE {
 ?rvr fp:country ?ct .
 FILTER (?ct, 'FR')
}
```

Q4
```
SELECT ?pdr
WHERE {
 ?pdr pr:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```

Q5
```
SELECT ?n
WHERE {
 ?rvr1 fp:name ?n .
 ?rvr1 fp:country ?ct .
 FILTER (?ct, 'FR')
}
```

Figura 35 – Consultas SPARQL sobre as ontologias de aplicação

Em seguida, cada subconsulta ilustrada na Figura 35 é reescrita com base nos mapeamentos locais da Figura 31, em termos de uma consulta sobre o esquema das fontes de dados apresentadas na Figura 36. O uso desses mapeamentos torna a reformulação simples e não-ambígua.

Q1'
```
SELECT ?t ?cm ?rt ?rvr ?pdr
WHERE {
 ?r rdf:type a:Review .
 ?r a:title ?cm .
 ?r a:rating ?rt .
 FILTER (?rt > 8 )
 ?r a:reviewer ?rvr .
 ?r a:reviewFor ?p .
 ?p rdfs:label ?t .
 ?p a:producer ?pdr .
}
```

Q2'
```
SELECT ?t ?cm ?rt ?n
WHERE {
 ?r rdf:type eb:ProductReview.
 ?r e:revTitle ?cm .
 ?r e:rate ?rt .
 FILTER (?rt > 8 )
 ?r e:reviewFor ?p .
 ?r e:revName ?n ;
 ?p rdfs:label ?t .
 ?p e: fabricator ?pdr .
 ?pdr e:country ?cy .
 FILTER (?cy, 'USA')
}
```

Q3'
```
SELECT ?rvr
WHERE {
 ?rvr f:country ?ct .
 FILTER (?ct, 'FR')
}
```

Q4'
```
SELECT ?pdr
WHERE {
 ?pdr p:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```

Q5'
```
SELECT ?n
WHERE {
 ?rvr1 f:name ?n .
 ?rvr1 f:country ?ct .
 FILTER (?ct, 'FR')
}
```

Figura 36 – Consultas SPARQL sobre as fontes de dados

6.4.1 O papel das ligações *same-as* na reformulação da consulta

Considerem-se as instâncias dos esquemas-fonte na Figura 37. A Figura 38 mostra as instâncias das ontologias de aplicação obtidas pela aplicação dos mapeamentos descrito na Figura 30. A Figura 39 ilustra um exemplo da ontologia de domínio *Sales* obtido pela aplicação dos mapeamentos de mediação da Figura 30.

A consulta SPARQL, na Figura 32, referindo-se às instâncias da ontologia de domínio na Figura 39, da consulta retorna apenas [*Apple iPad, 9.5, Wonderful Tablet*], mas a resposta correta deveria incluir também [*Netbook Sony Vaio, 9.0, Very fast!! Nice Item!*], porque, com base na ligação interontologia *(same-as)*, pode-se inferir que *rvr4* e *rvr2* são a mesma entidade (cabe lembrar que se diz que a ligação *same-as* é virtual porque derivada da aplicação de uma regra, como discutido na seção 5.2).

A Figura 35 exibe a consulta obtida do *unfolding* da consulta *Q* sobre as ontologias de aplicação, com base nos mapeamentos de mediação da Figura 30. Referindo-se aos dados do grafo na Figura 38, a consulta retorna somente [*Apple iPad, 9.5, Wonderful Tablet*]. Observe-se que a consulta *Q* aplicada ao grafo da Figura 39 fornece todas as respostas corretas: { [*Apple iPad, 9.5, Wonderful Tablet*], [*Netbook Sony Vaio, 9.0, Very fast!! Nice Item!*] }.

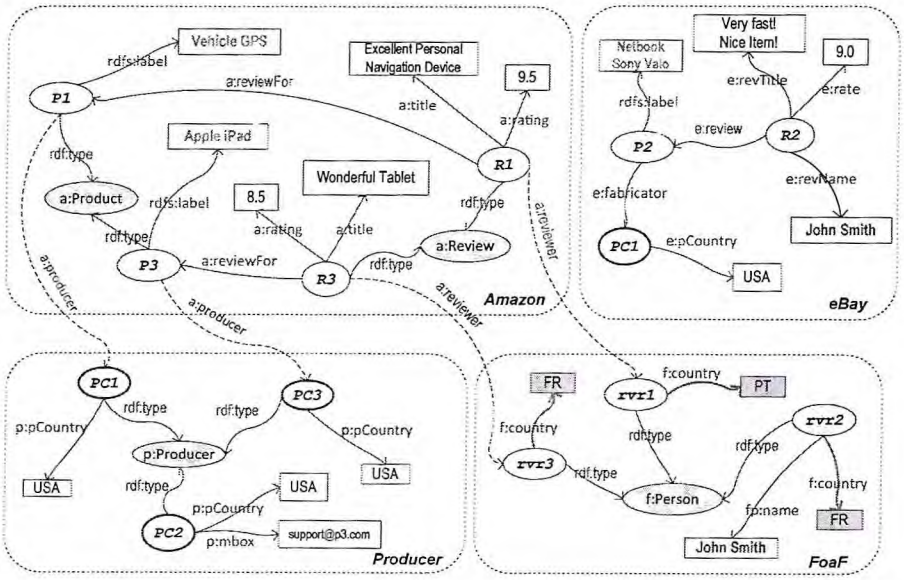


Figura 37 – Exemplo de dados RDF localizados em quatro diferentes fontes de dados: *Amazon*, *eBay*, *Producer* e *FoaF*

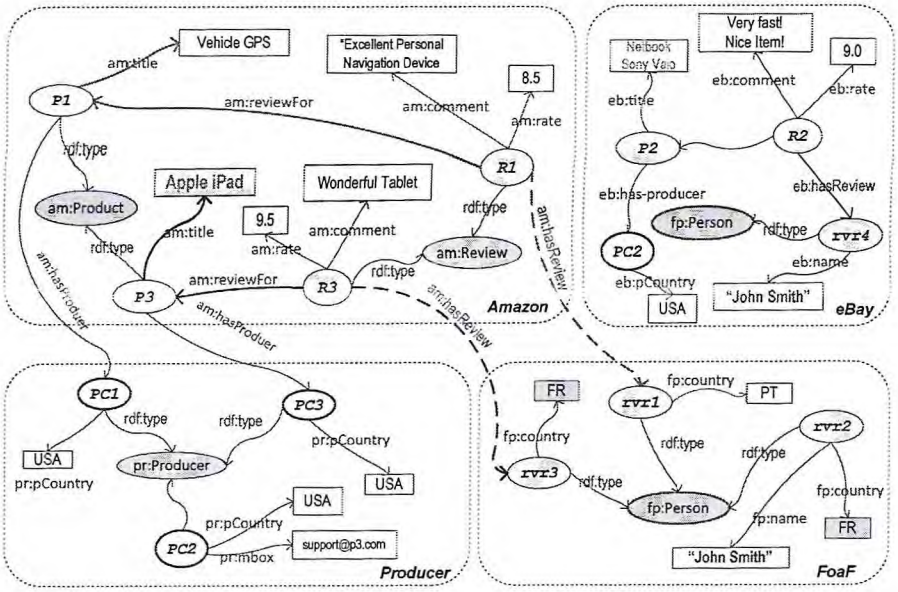


Figura 38 – Exemplo de dados RDF localizados em quatro diferentes ontologias de aplicação, designadas de *Amazon*, *eBay*, *Producer* e *FoaF*

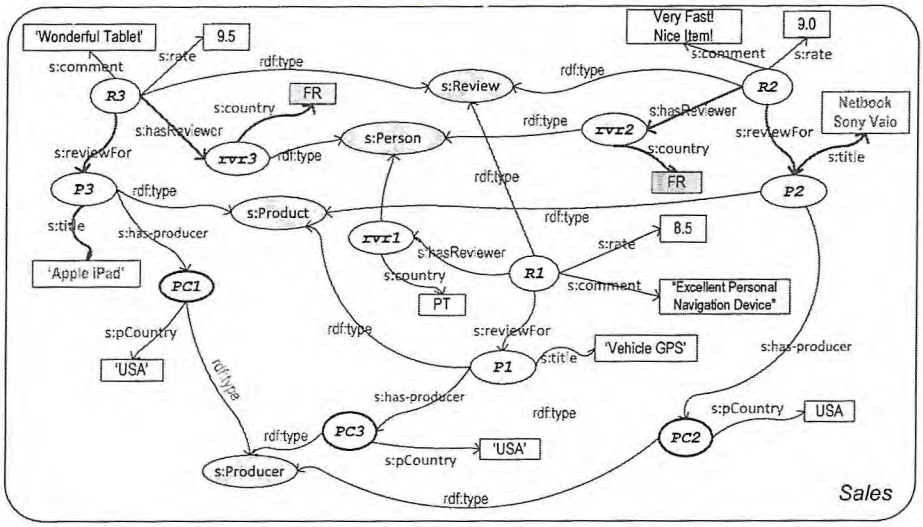


Figura 39 – Exemplo de dados RDF para ontologia de domínio *Sales*

Uma importante questão relacionada à integração de dados no padrão de *Linked Data* é que, na presença de ligações *same-as*, aumenta-se o espaço de busca de forma a obter melhores resultados.

6.5 Execução e consolidação des resultados

Cada subconsulta sobre uma fonte de dados gerada na etapa anterior é submetida ao SPARQL *endpoint* correspondente. Os resultados da subconsulta são retornados para o mediador, em que o resultado final é construído conforme o plano de execução. Detalhes sobre a otimização nos SPARQL *endpoints* estão fora do escopo deste trabalho, pois os mesmos são autônomos.

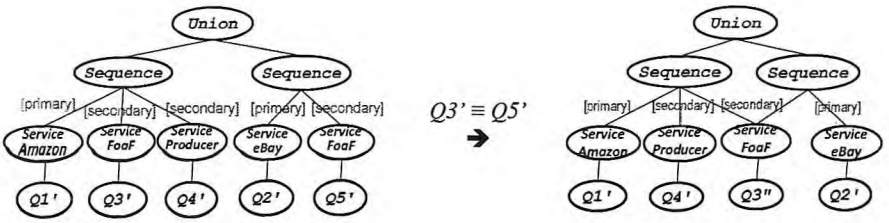
Em uma arquitetura de mediador, minimizar o tempo de resposta é, geralmente, mais importante do que maximizar o *throughput*. No entanto, o transporte de dados é muito caro, assim, o outro objetivo é minimizar a quantidade de dados transferidos dos SPARQL *endpoints* para o mediador.

Esta etapa gera o plano de execução, considerando os seguintes objetivos:

a) o potencial de ganho pelo processamento paralelo das subconsultas. É fornecida uma estratégia de paralelismo *pipelining*, baseado no algoritmo *pipelining hash-join* (WILSCHUT; APERS, 1993) para lidar com a natureza variável do acesso aos dados no padrão de *Linked Data* e produzir resultados o mais cedo possível;

b) o potencial de redução da transferência de dados de um SPARQL *endpoint* para o mediador, usando estratégia semelhante à de semijunção (HAAS et al, 1997). Foi proposto um operador de junção denominado *set-bind-join*, baseado no operador *bind-join* (RAJARAMAN; SAGIV; ULLMAN, 1995).

A seguir, explica-se a etapa de execução. A Figura 40(a) ilustra uma representação gráfica do plano de execução para as consultas apresentadas na Figura 36, as consultas equivalentes são identificadas. Note-se que as consultas *Q3'* e *Q5'* são similares. Assim, o plano de execução é alterado, como se visualiza na Figura 40(b) gerando a nova consulta *Q3"*, que é apresentada na Figura 41(b), evitando-se, assim, buscar o mesmo dado mais de uma vez na mesma fonte de dados.



(a) (b)

Figura 40 – (a) plano de execução; (b) plano de execução equivalente eliminando a consulta duplicada *Q5'*

Neste exemplo, o operador de junção usa o algoritmo *set-bind-join* (detalhes desse algoritmo são apresentados na seção 8.4.2). Considerando que cada fonte de dados é capaz de responder a consultas SPARQL, o operador *set-bind-join* processa segundo uma estratégia em que os dados resultantes de uma das consultas (tuplas) são lidos primeiro, e os valores correspondentes as variáveis de junção (*binds*) dessa consulta são enviados *set-a-set* encapsulados na cláusula *BINDINGS* da outra consulta SPARQL que participa da operação de junção.

Esse algoritmo consiste, essencialmente, de três etapas:

a) **Scan –** as *fontes de dados* são acessadas uma única vez e em paralelo, para obter todas as propriedades de junção (*binds*) e outros valores que contribuem para a resposta com eliminação de duplicatas. Na consulta *Q3"* (Figura 41-a), são projetadas instâncias da variável *name* (*?n*) e o *URI-link* (*?rvr*) dos revisores localizado na França '*Fr*'. A consulta *Q4'* recupera os fabricantes (*?pdr*) dos Estados Unidos '*USA*'.

Q3"
```
SELECT distinct ?n ?rvr
WHERE {
 ?rvr f:name ?n .
 ?rvr f:country ?ct .
 FILTER(?ct, 'FR')
}
```
(a)

Q4'
```
SELECT distinct ?pdr
WHERE {
 ?pdr p:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```
(b)

Figura 41 – Consultas SPARQL sobre as fontes de dados *Producer* e *Foaf*

Supondo-se que se obtiveram os seguintes resultados das consultas da Figura 41:
Q3"(?n, ?rvr): { (n1, r1), (n2, r2)}
Q4'(?pdr) : { p1, p2 }

b) **Processamento set-binds –** Esta etapa visa à redução do tamanho dos resultados intermédios e a diminuição da sobrecarga no mediador e pode ser subdividida em

dois passos:

i) executam-se, em paralelo, as outras consultas SPARQL para cada subconjunto de *binds* obtido na primeira etapa. A cláusula *BINDINGS* é usada para restringir os dados que devem ser recuperados. Note-se que as subconsultas *Q1'* e *Q2'* são reescritas nas subconsultas *Q1"* e *Q2"* (vide Figura 42) adicionando-se os *bindings* obtidos previamente das consultas *Q3"* e *Q4'*.

Q1"

```
SELECT ?t ?cm ?rt
WHERE {
 ?r rdf:type a:Review .
 ?r a:title ?cm .
 ?r a:rating ?rt .
 FILTER (?rt > 8 )
 ?r a:reviewer ?rvr .
 ?r a:reviewFor ?p .
 ?p rdfs:label ?t .
 ?p a:producer ?pdr.
} BINDINGS ?rvr, ?pdr {
 (r1, p1), (r1, p2)
 (r2, p1), (r2, p2)
}
```

Q2"

```
SELECT ?t ?cm ?rt
WHERE {
 ?r rdf:type
 eb:ProductReview.
 ?r e:revTitle ?cm .
 ?r e:rate ?rt .
 FILTER (?rt > 8 )
 ?r e:reviewFor ?p .
 ?r e:revName ?n .
 ?p rdfs:label ?t .
 ?p e: fabricator ?pdr .
 ?pdr e:country ?cy .
 FILTER (?cy, 'USA')
} BINDINGS ?n {
 (n1, n2)
}
```

Figura 42 – Consultas sobre as ontologias-fonte *Amazon* e *eBay* incluindo a cláusula *BINDINGS*

Os resultados intermediários de *Q1"* e *Q2"* são filtrados com os *binds* transmitidos e apenas um conjunto de resultados relevantes é retornado. Assim, o mediador receberá menos resultados intermediários, o que diminui o custo de transmissão e a sobrecarga de processamento no mediador. Para este exemplo o conjunto de *binds* foi enviado numa única consulta.

ii) transmite ao mediador os resultados das consultas *Q1"* e *Q2"* de maneira paralela e assíncrona.

c) **Junção no mediador –** define a execução da junção no mediador (*inner-join*). Para este exemplo, esta etapa não será necessária, pois o resultado das consultas *Q3"* e *Q4"* é formado somente por *binds*, que são utilizados apenas para restringir os resultados de *Q1"* e *Q2"*, e não contribui diretamente para a resposta do usuário com outros valores.

Por fim, é feita a união dos resultados de *Q1"* e *Q2"* para compor o resultado final em conformidade com o plano gerado na etapa anterior.

7 REFORMULAÇÃO DE CONSULTA

Neste capítulo, primeiro, apresenta-se uma introdução ao problema de reformulação de consulta. Em seguida, na seção 7.2 discute-se a caracterização do problema, isto é, a utilização de propriedades e ligações interontologias durante a reformulação da consulta para obter resultados mais completos. Na seção 7.3, apresenta-se o algoritmo de reformulação de consulta que usa as propriedades e ligações interontologias para reescrever a consulta sobre as ontologias de aplicação e também realizar a tradução de cada subconsulta sob a respectiva fonte de dados. Na seção 7.4 apresentam-se as regras de correção para validar o algoritmo proposto. Por fim, na seção 7.5 mostra-se um sumário do capítulo.

7.1 Introdução

Em um sistema de integração de dados baseado em mediadores, consultas são efetuadas sobre o esquema de mediação e não sobre os esquemas das fontes de dados. As consultas sobre o esquema de mediação devem ser decompostas em subconsultas direcionadas às fontes de dados. Portanto, o sistema deve possuir mecanismos para reformulação de consultas feitas sobre o esquema de mediação em subconsultas sobre os esquemas das fontes e garantir que a reformulação esteja correta, ou seja, as respostas e resultados obtidos das fontes de dados correspondem à resposta correta para determinada consulta sobre o esquema de mediação.O sistema também deve garantir que fontes com dados irrelevantes para a consulta não sejam acessadas e, assim, evitar o acesso desnecessário a uma fonte de dados.

Segundo Halevy et al. (2006), o processo de reformulação de uma consulta pode ser dividido em duas etapas: a reescrita da consulta – que gera uma expressão de consulta; e a resolução da consulta – cujo resultado é o conjunto das respostas possíveis para aquela expressão de consulta. Para viabilizar as duas etapas é necessário fazer uso de mapeamentos entre os esquemas. Os mapeamentos descrevem relacionamentos entre os termos usados em dois ou mais esquemas. Isso significa que soluções para o problema de reformulação de consulta devem incluir linguagens para especificação de mapeamentos e algoritmos que usem esses mapeamentos para resolver ou responder adequadamente às consultas.

7.2 Algoritmo de reformulação de consulta

O objetivo geral do algoritmo de reformulação da consulta é construir um plano de consulta que inclua somente subconsultas para as fontes de dados que podem contribuir com qualquer resultado intermediário. A topologia deste planos de execução é uma árvore onde as folhas são fontes de dados e os nós são operadores algébricos.

As ontologias de aplicação que serão integradas por essas operações representam fragmentos homogêneos da ontologia de domínio.

Os operadores de integração utilizados na geração do plano: *Service*, *Union* e *Sequence*:

a) o operador *Service* é utilizado para identificar a URI do SPARQL *endpoint* a que será enviada uma subconsulta. Este operador foi definido recentemente (BUIL-ARANDA; ARENAS; CORCHO, 2011) para ser utilizado na formulação de consultas federadas do *SPARQL 1.1* e está definido no engine Jena/ARQ;

b) o operador *Union* é definido na álgebra SPARQL (PÉREZ; ARENAS; GUTIERREZ, 2009), neste trabalho, sua utilização é trivial, apenas se faz a concatenação de resultados obtidos provenientes de consultas nas fontes de dados ou de resultados de outra operação de integração;

c) o operador *Sequence* foi desenvolvido no Jena/ARQ especificamente para executar junções múltiplas, em que as operações de junção são combinadas em uma única sequência, que pode ter *n* suboperadores. Na implementação do Jena/ARQ, este operador realiza uma abordagem de junção *left-deep,* enquanto no SemWIQ (LANGEGGER, 2010), que também utiliza este operador, é feita a reordenação das junções com base em estatísticas das fontes de dados.

A Figura 43 ilustra o algoritmo de reformulação de consulta, cujos principais passos são como segue.

a) Primeiro, selecionam-se as ontologias de aplicação cujo vocabulário contém o conceito primário da árvore de consulta (*QT*). Serão denominadas ontologias de aplicação primárias.

b) Então, para cada vocabulário *V* da ontologia primária, é gerada a árvore da consulta reformulada (*RQT*), utilizando o procedimento recursivo *REWRITE_NODE* apresentado na Figura 43. Esse procedimento reescreve cada propriedade de uma árvore de consulta (*QT*) em termos de *V*, ou utiliza propriedades e ligações interontologia para descobrir outros vocabulários e reescrever as propriedades que não estão em *V*. Para cada novo vocabulário encontrado, cria-se uma subconsulta sobre uma ontologia de aplicação denominada ontologia secundária. Por fim, o operador *Sequence* é adicionado como nó-pai para identificar as operações de junção sobre as subconsultas sobre a ontologia primária e suas ontologias secundárias.

c) O plano final da consulta (*FQP*) consiste na união dos planos de consulta reformulada nas ontologias primárias. Caso tenha sido identificada mais de uma ontologia primária, o operador *Union* será o nó-raiz.

O exemplo que segue ilustra como o algoritmo reformula uma árvore de consulta. Considere a árvore de consulta (*QT*) da Figura 33. O conceito primário é *s:Review*, que ocorre no vocabulário das ontologias de aplicação *Amazon* e *eBay*. Para gerar o *RQT* para uma ontologia primária, o algoritmo chama o procedimento *REWRITE_NODE* com dois parâmetros: uma árvore de consulta *QT*, com nó-raiz *r*, e o vocabulário da ontologia primária. Esse procedimento visita recursivamente cada nó *N* de *QT* e reescreve cada propriedade *p* de *N*. O procedimento considera três diferentes tipos de propriedade: *a*, *b* e *c*, a seguir.

a) Quando *p* está no vocabulário de *V* e *p* não é uma propriedade ou ligação *interontologia* (linha 5), então, substitui-se o espaço de nomes de *p* por *V*. Por exemplo, na Figura 44, a propriedade *s:comment* é substituída por *am:comment*.

b) Quando *p* está no vocabulário de *V* e *p* é uma propriedade cujo contradomínio de *p* está no vocabulário de *Vt*, tal que $Vt \neq V$ (linha 7-10), então, uma árvore de consulta secundária *SQT* é criada com nó-raiz *rnew*, que é uma cópia do nó *rnext*, que ocorre em *p*; os nós-filhos de *rnew* são todos os descendentes de *rnext*. Em seguida, a árvore de consulta *SQT* é reescrita usando espaço de nomes de *Vt*. No exemplo, a propriedade *s:hasReviewer* é uma propriedade *interontologia* (*am:Review*, *fp:Person*, *am:hasReviewer*). A Figura 44 sintetiza os principais passos para reescrever a propriedade *s:hasReviewer*: 1) cria-se a árvore de consulta secundária *SQT*; 2) apagam-se todos os nós descendentes de *?rvr* da árvore de consulta primária; 3) reescreve-se *SQT* com espaço de nomes de *FoaF* (*fp*). A variável de junção *?rvr*

da propriedade *am:hasReviewer* é utilizada na árvore *SQT* para implantar a junção (usando *URI-link*) entre as duas consultas. Observa-se, também na Figura 44, que procedimento semelhante ocorre para o *URI-link am:has-producer*.

REWRITE_NODE (QT, r, V)

Reescreve as propriedades (arestas) de um nó r em uma árvore de consulta (*QT*) com o vocabulário V e usa as ligações e propriedades interontologia para descobrir outros vocabulários e reescrever as propriedades que não estão em V. O resultado do procedimento REWRITE_NODE é uma árvore da consulta reformulada, cujo nó-raiz é um nó de junção com os seguintes nós-filhos: a árvore de consulta primária (árvore de consulta reescrita com o vocabulário V) e todas as árvores da consulta reescrita com outros vocabulários para as propriedades que não estão em V.

```
input:
   root node r of a query tree QT
   vocabulary V
output:
   reformulated query tree RQT
1    Begin
2     SS← ∅; RQT ← QT;
4     for each edge e(r, rnext) with label pe whose range is in V do
5       replace the label pe of e by V:pe
6       if pe is an inter-ontology link whose range is in Vt such that
        Vt ≠ V
7          createUriLinkSQT(r); rnew ← rnext
9          rewrite_Node (QT, rnew, Vt)
10          add rnew to SS
11      Else
12         rewrite_Node (QT, rnext, Vt)
13     end-if
14    end- or
15    let E={label(e) | e is an edge that leaves node r and label (e)∉ V}
16    for each Vocabulary Vi such that there exists a virtual same-as link
      (V:cr, Vi:cr, PJ) such that E ∩ Vi ≠ ∅
17       createSame-asLinkSQT (r); rnew ← rnext
18       rewrite_Node(QT, rnew, Vi)
19       add rnew to SS
20   end-for
21   if SS≠ ∅
22     add r to SS
23     RQT ← SS
24    add root_node_Service(getURL(V));
25   end-if
26   End
```

Figura 43 – Algoritmo de reformulação de consulta

c) Se *p* não está no vocabulário de *V* (linhas 16-20), então, para cada ligação *interontologia same-as* (*V:cr, Vi:cr, PJ*) em *V*, cria-se uma árvore secundária de consulta *SQT*, que é uma cópia do subgrafo contendo o nó *N* e todos os descendentes das propriedades que não ocorrem em *V*. Em seguida, reescreva *SQT* com *Vi*.

Por exemplo, a propriedade *country* não está no vocabulário de *eBay*. Mas a utilização de ligação *interontologia same-as* (*eb:Person*, *fp:Person*, *name*), possibilita juntar instâncias de *eb:Person* com instâncias de *fp:Person* para obter o país que está no vocabulário de *FoaF*.

A Figura 45 sintetiza os principais passos para reescrever a propriedade *s:country*: 1) cria-se a árvore de consulta secundária *SQT* ilustrada na Figura 45; 2) exclui-se a aresta *country* da árvore de consulta primária; 3) reescreve-se a *SQT* com espaço de nomes FoaF (*fp*). A Figura 45 mostra o grafo da consulta secundário para a ligação *interontologia same-as* (*eb:Person*; *fp:Person*; *name*).

O resultado do algoritmo *REWRITE_NODE(N, V)* é uma árvore de consulta reformulada (*RQT*) cujo nó-raiz de cada subconsulta é o operador *SERVICE* (linha 24), que indica o SPARQL *endpoint* que será consultado. O operador *Sequence* corresponde à junção com os seguintes nós-filhos: a árvore de consulta primária (árvore de consulta reescrita com o vocabulário *V*) e todas as árvores de consultas secundárias geradas a partir das propriedades e ligações interontologia.

A Figura 44 exibe a árvore de consulta reformulada para o vocabulário de *Amazon*. A variável de junção *?rvr* da propriedade *am:hasReviewer* é utilizada na árvore de consulta secundária para implantar a operação de junção. A Figura 45 mostra a árvore de consulta reformulada para o vocabulário de *eBay*. A variável *?n* correspondente às propriedades *eb:name e fp:name*, que são obrigatórias (*bound*) em ambas as árvores de consulta para o processamento da operação de junção.

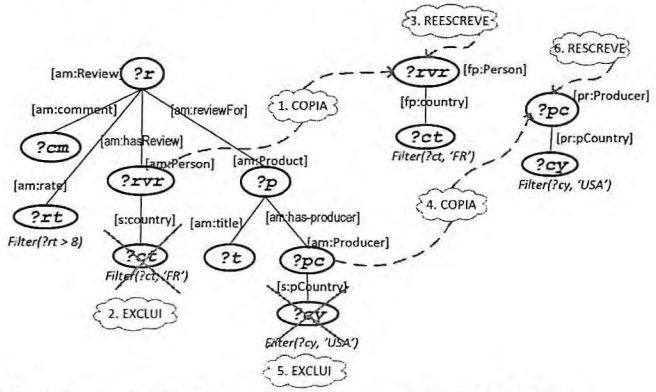


Figura 44 – Processamento de propriedades interontologia *URI-links*

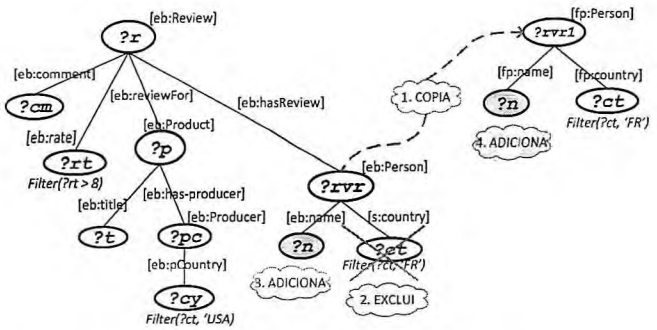


Figura 45 – Processamento de ligações interontologia *same-as*

Por fim, todas as árvores de consulta são agrupadas em um único plano de consulta final (FQP). A Figura 46 ilustra o FQP gerado no exemplo. Note-se que este plano de execução gera cinco subconsultas que devem ser submetidas a quatro ontologias de aplicação.

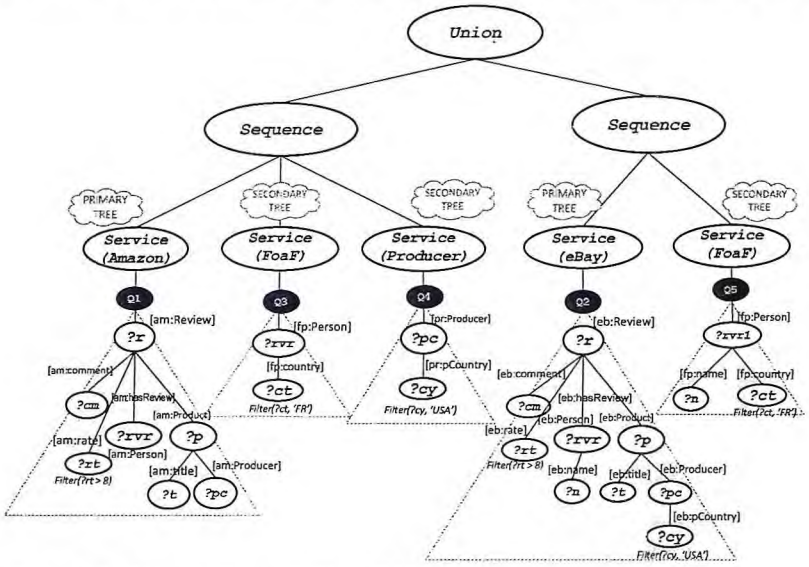


Figura 46 – Plano de Execução Final (FQP)

Em seguida, cada subconsulta é traduzida, com base nos mapeamentos locais, em termos do esquema das fontes de dados. O uso desses mapeamentos torna a tradução de cada subconsulta simples e não-ambígua, conforme formalizado por Sacramento et al. (2010).

7.3 Considerações do capítulo

A principal contribuição deste capítulo é um algoritmo de reformulação de consulta, em que uma consulta SPARQL Q submetida a uma ontologia de domínio é reescrita em um conjunto de subconsultas Q_i sobre múltiplas fontes de dados representadas por um plano de execução de consulta. Dois diferenciais deste algoritmo são: a) busca dados apenas em fontes de dados que podem contribuir com qualquer resultado intermediário, sem precisar recorrer a mecanismos de inferência para fazer a expansão da consulta; b) utiliza propriedades e ligações interontologias, que é um diferencial importante deste trabalho, pois, como foi enunciado anteriormente, torna possível obterem-se dados de uma fonte de dados que pode ser complementada por dados de outras fontes.

Os mapeamento são fundamentais nesta etapa, pois para cada conceito ou propriedade ($OD{:}C_i$ ou $OD{:}P_J$) na ontologia de domínio deve existir um mapeamento de mediação (γ) que mapeie em pelo menos um conceito ou propriedade ($OA_K{:}C_i$ ou $OA_K{:}P_J$) de uma ontologia de aplicação (vide **Definição** 9). Em virtude desses mapeamentos de mediação, a reformulação de consulta sobre a ontologia de domínio em subconsultas sobre as ontologias de aplicação pode ser feita automaticamente, sem precisar recorrer a nenhum mecanismo de inferência, esses mapeamentos, também, garantem que fontes de dados irrelevantes para a consulta não sejam acessadas e, desse modo, evitem-se acessos desnecessários às fontes de dados. Enquanto os mapeamentos locais garantem a tradução correta de cada subconsulta sobre a ontologia de aplicação traduzida em termos da respectiva fonte de dados. Libkin e Sirangelo (2011) formalizaram esse problema no contexto de *data exchange*.

É importante destacar que o plano gerado nesta etapa pode ser executado diretamente sobre o *engine* Jena/ARQ, que suporta consultas federadas formuladas como expressão algébrica. Isso demonstra a independência entre as etapas de reformulação e execução do método apresentado no Capítulo 6. No Capítulo 8, a seguir, apresenta-se a estratégia de execução de consultas distribuídas proposta e no Capítulo 9 analisam-se os resultados experimentais que evidenciam o melhor desempenho na estratégia de execução proposta em relação às consultas federadas no *engine* Jena/ARQ.

8 EXECUÇÃO DE CONSULTAS SPARQL DISTRIBUÍDAS PARA INTEGRAÇÃO DE DADOS

O suporte para execução do plano gerado na etapa anterior e a consolidação dos resultados envolvem o processamento distribuído de consulta em SPARQL *endpoints* autônomos para integração de dados. Neste sentido, este Capítulo apresenta a estratégia de execução adotada neste trabalho. Inicialmente, são apresentados alguns conceitos da área de execução de consultas distribuídas. Na seção 8.1, discute-se uma motivação para estudo do problema. Na seção 8.2 expõem a discussão do problema. Na seção 8.3 são apresentadas algumas abstrações denominada de características de execução. Na seção 8.4 apresentam-se os algoritmos de junção que exploram a natureza dos *Linked Data*. A seção 8.5 trata da estratégia adaptativa que permite uma otimização em tempo de execução, em contraste com o processo de otimização baseado em uma função de custo, que é estático.

8.1 Introdução

Os processos de execução de consultas SPARQL têm recebido cada vez mais atenção da comunidade científica (ATRE et al, 2010; HARTIG; BIZER; FREYTAG, 2009; GROPPE eta al, 2009; NEUMANN; WEIKUM, 2010) devido à crescente necessidade de gerenciar fontes de dados com bilhões de triplas (*triples Challenge* 2011[14]). Porém a execução de consultas SPARQL distribuídas ainda está em estágio inicial e indica a necessidade de uma base tecnológica escalável em um espaço global de dados interligado constituído de bilhões de triplas RDF, que não se limite ao processamento de consultas SPARQL locais (OLAF GORLITZ et al., 2011).

Com isto em mente, a execução do plano gerado pelo algoritmo de reformulação (vide Capítulo 7) segue as premissas a seguir.

> **Aumento da vazão (*throughput*) e diminuição do tempo de resposta obtido pela paralelização do processamento de subconsultas** – é fornecida uma estratégia de execução de consulta paralela que utiliza o algoritmo *pipelining*

[14] http://challenge.semanticweb.org/

hash-join (WILSCHUT; APERS, 1993) para lidar com a imprevisibilidade no acesso aos SPARQL *endpoints* (seção 8.4.1). Esse algoritmo possui duas características relevantes: a) produzir resultados incrementalmente à medida que ficam disponíveis ainda que tenha que continuar com o processamento de outras tuplas; b) permitir execução contínua do algoritmo mesmo quando ambos SPARQL *endpoints* estiverem em atraso, apesar da redução dos resultados parciais gerados.

Redução do volume de dados transmitido ao mediador – é definido um algoritmo de junção *set-bind-join* (vide seção 8.4.2) que objetiva a eliminação previamente de dados que não contribuem para resposta e explora o paralelismo intraoperador visando melhorar a eficiência de execução. Este algoritmo tem objetivo semelhante ao operador de *semijunção* (KOSSMANN, 2000) do modelo relacional e seu funcionamento baseado no operador *bind-join* (HAAS, 1997; RAJARAMAN; SAGIV; ULLMAN, 1995).

Idealmente, a execução deve reagir às mudanças no ambiente – a proposta é uma estratégia de execução em que a execução se adapte tanto a atrasos na chegada dos dados provenientes dos SPARQL *endpoints* quanto ao volume de resultados intermediários que chegam ao mediador visando melhorar o desempenho durante o processamento. No enfoque deste estudo, a adaptação é acionada somente quando necessário.

Minimização do tempo de resposta – inspirado no protocolo HTTP usado pelos SPARQL *endpoints*, que apresentam resultados assim que chegam, em vez de esperar que os dados sejam recebidos por completo. A estratégia de execução deste estudo também permite minimizar o tempo para obter os primeiros resultados. Explora-se um modelo de notificação de eventos baseado no padrão de projeto *observer* (GAMMA et al., 1995), que permite propagar os dados automaticamente a medida que seja produzidos.

A Figura 47 ilustra as etapas do processamento de consulta adotadas neste trabalho. No passo 1, o algoritmo de reformulação (vide seção 7.3) gera um plano de execução que representa uma consulta SPARQL federada; no passo 2 é gerado um plano

inicial (seção 8.5.1); e no passo 3 é realizado o processamento adaptativo e a integração dos resultados (seção 8.5.2), em que se monitora, durante a execução, a quantidade de dados recuperados e atrasos na entrega dos dados.

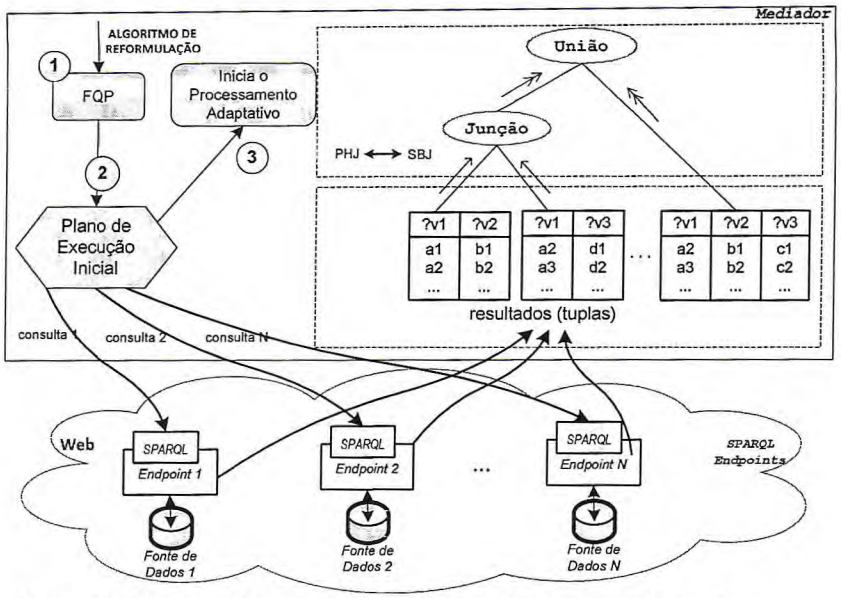


Figura 47 – Estratégia de execução com adaptação dos algoritmos de junção

8.2 Discussão do problema

Nesta seção é detalhado o problema de processamento distribuído de consulta motivado pela integração de dados no padrão de *Linked Data*. Sejam as fontes de dados: *eBay(title, name, hasReviewer, reviewFor, has-producer), FoaF(name, mbox, country)* e *Producer(pName, pCountry),* usando, respectivamente, os seguintes prefixos "*eb:*", "*pr:*" e "*fp:*" (vide Figura 29) para distinguir os espaços de nomes:.

Observam-se as subconsultas SPARQL *Q1* (*eBay*), *Q2* (*Foaf*) e *Q3* (*Producer*), apresentadas na Figura 50, que recuperam os títulos (*?t*) dos produtos de fabricante dos Estados Unidos (*USA*) avaliados por um consumidor com *email* '*john@gmail.com*'.

Q_1
```
SELECT ?t ?n ?pdr
WHERE {
 ?r eb:hasReviewer ?rvr .
 ?rvr eb:name ?n .
 ?r eb:reviewFor ?p .
 ?p eb:title ?t .
 ?p eb:has-producer ?pdr .
}
```

Q_2
```
SELECT ?n
WHERE {
 ?rvr1 fp:name ?n .
 ?rvr1 fp:mbox ?m .
 FILTER(?m,'john@gmail.com')
}
```

Q_3
```
SELECT ?pdr
WHERE {
 ?pdr pr:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```

Figura 48 – Subconsultas sobre as ontologias de aplicação *eBay*, *Person* e *Producer*

O filtro *FILTER(?m, 'john@gmail.com')* exemplifica a necessidade do fornecimento de dados publicados unicamente pela fonte de dados *FoaF*. Neste caso, talvez seja desejável obter previamente o nome do revisor. Comportamento similar ocorre no filtro (cy='*USA*') referente à subconsulta da fonte de dados *Producer*.

A partir dessas informações, pode-se definir uma sincronização entre as subconsultas que irão compor o plano de execução. Na Figura 49 há algumas possíveis ordens de execução de Q_1, Q_2 e Q_3. Por exemplo, na Figura 49(a), primeiro obtém-se em paralelo os fabricantes dos Estados Unidos concomitante com revisor com email '*john@gmail.com*', que serão enviados como *binds* na consulta Q_1; já na Figura 49(b) todas as três subconsultas são executadas em paralelo, e a integração é feita no mediador; enquanto na Figura 49(c) se executa primeiro Q_1, que recupera o nome dos revisores de qualquer país juntamente com os *URI-links* de todos os fabricantes.

É importante ressaltar que, nos cenários da Figura 49 (b) e (c), a subconsulta Q_1 poderá retornar grande volume de dados intermediários já que nenhuma restrição é aplicada Por fim, na Figura 49(d), primeiro obtém-se em paralelo os fabricantes dos Estados Unidos e o revisor com email '*john@gmail.com*', porém a subconsulta Q_1 aguarda somente pelo resultado de Q_2, que será enviada como *bind* na consulta Q_1, e a junção desse resultado com Q_3 será processada no mediador.

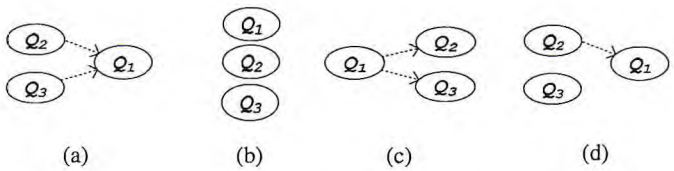


Figura 49 – Algumas possíveis ordens de execução das consultas *Q1*, *Q2* e *Q3*

Observa-se que determinar a melhor ordem de execução das subconsultas não é uma tarefa fácil. Quanto ao processamento de consulta, destacam-se três problemas que devem ser considerados.

informações estatísticas são escassas – a autonomia dos SPARQL *endpoints*, que são públicos e acessados livremente por qualquer usuário ou aplicação na Web com uma simples requisição HTTP GET ou POST, e a facilidade de atualização de dados publicados no padrão de *Linked Data* torna difícil obter e manter informações estatísticas de qualidade sobre os dados disponibilizados. Essas características fazem da etapa de otimização baseada em funções de custo inadequada para o domínio deste trabalho. Outro ponto-chave é que a quantidade de dados no padrão de *Linked Data* tem crescido sobremaneira nos últimos anos, e não se veem sinais de que essa tendência reverter-se-a em um futuro próximo. O crescimento é decorrência da publicação de dados abertos disponíveis livremente que podem ser consumidos por qualquer aplicação via SPARQL *endpoint*.

Tempo de resposta de uma consulta e a quantidade de dados recuperados – os SPARQL *endpoints* podem apresentar problemas de desempenho frente à situações características do domínio Web. Os mesmos estão sujeitos a variações no tempo de resposta da consulta e a quantidade de dados retornados é desconhecida, e o desempenho da rede pode degradar-se de forma inesperada. Por exemplo, o SPARQL *endpoint* pode ficar sobrecarregado atendendo a outras solicitações em um *momento de pico*, que se concentram muitos acessos, podendo tornar o plano ineficiente. Nesse caso, o ideal seria modificar o plano inicial de execução para outro mais eficiente, que é chamado de processamento adaptativo de consultas.

Operadores com mais de uma fase que não apresentam suporte à técnica de *pipelining* – por exemplo, o operador *hash-join* que possui as fases de construção e comparação e não usa *pipelining*; isto é, a segunda fase só se inicia quando a primeira fase é totalmente concluída. Logo, a fase de comparação somente é executada após a fase de construção ser totalmente concluída, o que pode afetar seu desempenho no processamento, caso ocorra atraso na entrega dos dados na primeira fase.

Na próxima seção, discutem-se as técnicas usadas na estratégia de execução, e na sequência, os algoritmos de junção e a estratégia adaptativa que visa a superar os problemas supracitados.

8.3 Características de Execução

A estratégia de execução é determinada pela combinação de características que atendam as premissas apresentadas na seção 8.1. É possível classificar tais características em três grupos principais: execução paralela de consultas, modelo baseado em eventos, e técnicas de entrega de dados. A seguir, detalham-se cada um desses grupos.

8.3.1 Execução paralela de consultas

As técnicas de paralelização para operações de junção utilizadas neste trabalho estão diretamente relacionadas ao paralelismo empregado em âmbito intraoperador e interoperador, como explicados a seguir.

Interoperador – os vários operadores de uma consulta podem ser executados em paralelo utilizando a independência entre eles. É possível dividir o emprego de paralelismo interoperador em dois grupos (GRAEFE, 1993): a) *horizontal* (execução paralela), paralelismo entre operadores independentes entre si, ou seja, o resultado de um operador não é entrada para o outro; b) *vertical* (execução *pipelining*), paralelismo entre operadores dependentes entre si, em que o resultado de um é entrada para o outro. Esse tipo de paralelismo é adotado pelo algoritmo de junção *pipelining hash-join*.

Intraoperador – isolando uma subconsulta, pode-se realizar seu processamento em paralelo executando o operador (algoritmo) em *threads* distintas (GRAEFE, 1993), com diferentes conjuntos de dados. Esse tipo de paralelismo é adotado pelo algoritmo de junção *set-bind-join*, proposto neste trabalho.

Na sequência, são exemplificados os tipos de paralelismo, lembrando que uma das premissas é produzir os resultados da consulta do usuário o mais breve possível. Considere-se, então, o plano de execução da Figura 50, em que os nós-folha representam as consultas sobre os SPARQL *endpoints* autônomos, os nós-não-folha representam as operações de integração e as arestas (arcos) os dados transmitidos entre os operadores na forma produtor-consumidor, enquanto as setas indicam o fluxo dos dados.

Primeiramente, tem-se que as junções 1 e 2 podem ser executadas em paralelo (paralelismo interoperador horizontal), independentemente dos dados que consomem. Verifica-se que os dados produzidos pelas junções 1 e 2 serão consumidos na operação de união que caracteriza o paralelismo interoperador vertical, ou simplesmente, paralelismo *pipelining* (vide Figura 50a).

O paralelismo *pipelining* consiste em estabelecer um fluxo de dados entre os operadores, em que um nó-filho é o *produtor* dos dados e o nó-pai é o *consumidor*. A grande vantagem do *pipelining* é que os dados não necessitam ser armazenados (*materializados*) temporariamente entre uma operação e outra. Será adotada, neste trabalho, a nomenclatura *produtor-consumidor*, em que o primeiro representa o operador que produz os dados e o segundo corresponde ao operador que consome os dados produzidos. Um operador pode ser produtor e consumidor de dados ao mesmo tempo.

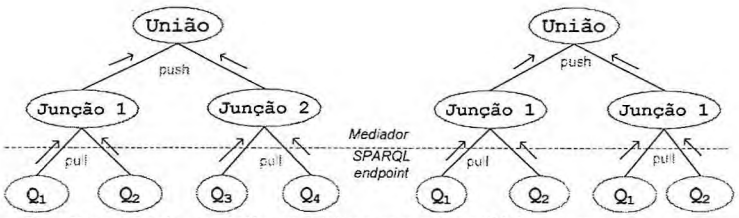


Figura 50 – Exemplo de paralelismo (a) interoperador e (b) intraoperador

A técnica de paralelismo intraoperador é ilustrada na Figura 50(b), o operador Junção1 foi duplicado em duas *threads*, que processam parte dos resultados das subconsultas Q_1 e Q_2.

8.3.2 Notificação dos operadores de integração usando um modelo baseado em eventos

O modelo baseado em eventos é muito conhecido na criação de interfaces gráficas com o usuário, onde os eventos gerados pelos componentes (botões, janelas, etc.) acionam código escrito para a aplicação específica.

Nesse modelo, um componente, conhecido como produtor, gera dados que são entregues a todos os componentes, conhecidos como consumidores, os quais demonstraram interesse em ser notificados (GAMMA et al., 1995). No contexto deste trabalho, tanto o produtor quanto o consumidor são operadores de integração de dados.

Um pré-requisito para a eficiência da estratégia de execução proposta neste trabalho está diretamente relacionado à velocidade com que os consumidores reagem ao receberem a notificação de um evento. O ideal é que a notificação dos consumidores possa ocorrer de forma automática, dessa maneira, gerando menos acoplamento.

Quando um resultado é gerado (no *produtor*), a notificação de evento propaga-se para todos os *consumidores* que registraram interesse em recebê-la. A propagação de um evento é executada assincronamente e, é completamente transparente ao produtor que gera o evento, o que possibilita ao consumidor executar sem estar ciente da existência do produtor. Tudo que deve conhecer é a estrutura de notificação de eventos que lhe interessa. Por conseguinte, os produtores e consumidores podem ser facilmente trocados ou reutilizados em outras situações, pois possuem um acoplamento fraco.

Para alcançar essas funcionalidades adota-se o padrão de projeto *observer* (GAMMA et al., 1995). Este padrão de projeto descreve como definir uma dependência entre objetos a fim de que, quando algum deles mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente.

8.3.3 Técnicas de entrega de dados

Outro ponto importante refere-se à técnica utilizada para transferência de dados entre os operadores. Adota-se o modelo híbrido descrito em Ozsu e Valduriez (1999), o qual

combina os mecanismos de *pull* (*puxar*) e *push* (*empurrar*). Nesse sentido, apresentam-se, a seguir, as técnicas utilizadas neste trabalho.

> **No mecanismo de entrega de dados baseada em *pull***, a transferência de dados do SPARQL *endpoint* (produtor) para o mediador (consumidor) é iniciada por um *pull* (*puxão*). Quando uma solicitação (consulta SPARQL) submetida pelo mediador é recebida por um SPARQL *endpoint*, o mesmo responde enviando os dados solicitados, ou seja, o mediador é que decide quando buscar os dados. A principal característica da entrega de dados baseada em *pull* é que a chegada de itens de dados é executada sem notificação ao mediador, que é equivalente ao método de acesso a páginas HTML na Web, e essa abordagem é mais adequada para lidar com a latência da rede, pois é dirigida por dados de entrada (OZSU; VALDURIEZ, 1999). Outra característica é que os SPARQL *endpoints* têm de ser continuamente interrompidos para lidar com solicitações de outros clientes. Além disso, os dados que um mediador pode obter do SPARQL *endpoint* são limitadas pelo momento (dados podem ser atualizados sem aviso prévio) e pelo conteúdo da solicitação (consulta SPARQL) do mediador.
>
> **O mecanismo de entrega baseado em *push*** ocorre apenas dentro do mediador, diferentemente do *pull*, o consumidor não precisa conhecer onde e quando procurar os dados porque os produtores geram os eventos e notificam imediatamente todos os consumidores interessados, cujo resultado produzido será a entrada para o próximo operador no plano (árvore) de execução. Com a estratégia de evento apresentada na seção 8.2.2, a meta é permitir que o consumidor receba as tuplas imediatamente.

Em síntese, os SPARQL *endpoints* oferecem a entrega baseada em *pull*, enquanto que a entrega de dados dentro do mediador é baseada em *push*. Na Figura 50, observa-se o modelo híbrido de entrega dos dados que ocorre de maneira *bottom-up,* como segue: a transferência de dados de um SPARQL *endpoint* para o mediador é baseada no modelo *pull*, em que os SPARQL *endpoints*, após a requisição do mediador, *entregam* os dados sem esperar que estes sejam solicitados. Vale ressaltar que, assim como *pull*, o *push* é feito seguindo uma ordem *bottom-up* no plano de execução, ou seja, a partir dos operadores correspondentes aos nós-filho da árvore de operador para operador do nó-raiz.

8.4 Algoritmos de junção que exploram a natureza dos *Linked Data*

A estrutura altamente distribuída e fragmentada dos dados no padrão de *Linked Data* apresenta desafios únicos quando se trata de execução distribuída de consultas. Primeiro, é preciso garantir a escalabilidade sobre um crescente conjunto de dados heterogêneos disponível em escala global. Segundo, dada a necessidade de juntar estruturas heterogêneas, é preciso tratar a existência de predicados de junção computacionalmente caros.

A operação de junção é a mais *cara* computacionalmente em tempo e espaço (GRAEFE, 1993) dentre as operações de integração, sendo que inúmeros trabalhos já foram publicados sobre o assunto e ainda é tema de discussão e pesquisa. São encontrados na literatura diversos algoritmos, grande parte deles baseados na utilização de laços aninhados e estratégias de *merge* e *hash* já consolidadas como alternativas viáveis em muitos bancos de dados comerciais.

Conforme já mencionado, pode-se perceber que alguns algoritmos de junção como o *merge-join* não permitem que resultados sejam produzidos antes que sejam consumidos todos os dados produzidos por uma das consultas, pois uma operação como *sort* causa parada no fluxo do *pipelining*. Além disso, estratégias de *hash* simples possuem duas fases de execução, sendo uma fase de construção, que deve receber todas as tuplas por completo, e uma fase de comparação, que só pode ser executada após a execução da fase anterior. O que leva os demais operadores a ficarem ociosos esperando por todos os resultados.

Devido a essas características, os algoritmos citados não são adequados para os requisitos de processamento de consultas SPARQL distribuídas. Precisam-se de algoritmos capazes de produzir resultados tão logo estejam disponíveis, e de oferecer bons resultados no processamento de consultas complexas mesmo em situações de imprevisibilidade, envolvendo junções com dados provenientes de vários SPARQL *endpoints* que são acessados globalmente sem nenhuma restrição de acesso. Nesse cenário, os dados podem chegar em qualquer ordem e com diferentes tempos de resposta, o que corrobora a dificuldade de prever os tempos de movimentação e resposta da consulta.

Algoritmos de junção precisam considerar a imprevisibilidade do tempo de resposta e, muitas vezes, com o desconhecimento do volume de dados recuperados. Alguns dos algoritmos de junção propostos pela comunidade de banco de dados distribuído/paralelo (GRAEFE, 1993; HAAS; et al, 1997; WILSCHUT; APERS, 1993) podem ser reutilizadas

para o processamento/otimização de consultas SPARQL para acesso a dados no padrão de *Linked Data*.

8.4.1 *Pipelining Hash-Join*

Este algoritmo foi projetado para permitir alto grau de paralelismo *pipelining* em sistemas de bancos de dados paralelos (WILSCHUT; APERS, 1993), também conhecido na literatura como *Symmetric Hash Join* (LAWRENCE, 2005). O acesso aos dados em um ambiente de *Linked Data* envolve grande número de fontes de dados acessados via SPARQL *endpoints* localizados remotamente, os quais são suscetíveis a sobrecargas, congestionamento e falhas, o que é típico do ambiente Web. Tais problemas podem resultar em atrasos significativos e imprevisíveis no acesso aos dados nas fontes de dados.

Como apresentado em Wilschut e Apers (1993), o algoritmo *pipelining hash-join* (PHJ) tem comportamento diferente do algoritmo de *hash-join*, que para iniciar a geração dos resultados demanda construir uma tabela *hash* inteiramente sobre uma das entradas, o que significa que nenhum resultado é retornado antes de pelo menos uma das tabelas ser completamente lida. Ou seja, *hash-join* é um operador de bloqueio e, como consequência, não é adequado para o processamento de dados no padrão de *Linked Data*.

Por outro lado, o PHJ é um algoritmo não bloqueante em que duas tabelas *hash* são construídas em paralelo ao ler os dados (*tuplas*) de dois SPARQL *endpoints* (vide Figura 51). Os resultados poderão ser produzidos em qualquer ponto da execução da junção, usando as informações parciais contidas nas tabelas *hashs*, sem ter de esperar o recebimento completo das tuplas de um SPARQL *endpoint*, como é o caso do *nested-loop-join* implementado no DARQ (QUILITZ; LESER, 2008). Devido a essas características, a operação de união é implementada seguindo a mesma estratégia do PHJ.

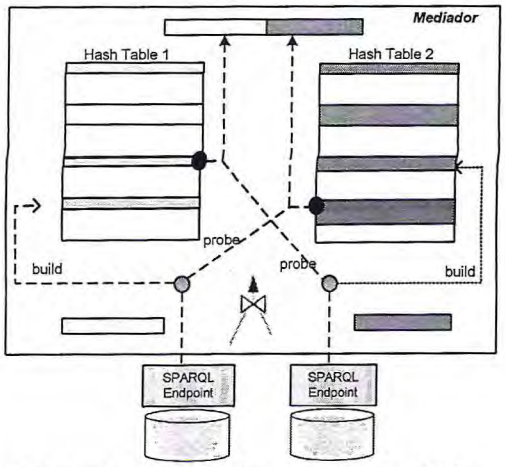


Figura 51 – Visão esquemática do *algoritmo pipelining hash-join*

Neste trabalho, a especificação do algoritmo PHJ, adaptado de Ozsu e Valduriez (1999), segue uma estratégia de paralelismo *pipelining* utilizando o modelo de iterador (GRAEFE, 1993), que define três funções principais: *open()*, *next()* e *close()*. Na Figura 54 tem-se o algoritmo que gerencia duas tabelas *hash*, uma para o resultado de cada consulta. Seu funcionamento é da seguinte forma: primeiro é chamado o método *open()*, que cria duas tabelas *hash* e as prepara para consumir os dados (tuplas) que serão transmitidos pelos SPARQL *endpoints* (linhas 7 a 10). Em seguida, executa-se o método *next()* (linhas 12 a 24), que recebe o resultado das consultas em um *stream* HTTP. Cada tupla (*t*) recebida é inserida na tabela de *hash* HT_k correspondente (ou seja, $HT_k = HT_1$ ou HT_2). Então, *t* é usado para sondar (*probe*) a tabela *hash* HT_j e efetuar as junções válidas. Por fim, é chamado o método *close()*(linha 26 e 27), que libera as tabelas *hash* alocados na memória.

Uma característica interessante deste algoritmo é que possibilita comunicar os resultados imediatamente ao próximo operador no plano de execução, chamando seu método *push* (linha 23). Os resultados são *push* imediatamente para o operador subsequente (consumidor) usando o modelo de notificação de eventos, o que permite a geração de resultados o mais cedo possível.

Uma desvantagem deste algoritmo consiste do fato de não tratar da redução da quantidade de dados transferidos dos SPARQL *endpoints* para o mediador. Ozakar et al. (2005) mostraram que esse custo de transferência pode chegar de 80% a 85% do custo total previsto da consulta.

```
1. Algoritmo: PHJ
2. Input: Hash tables HT1, HT2;
3. if( open() ) {
4.    next();
5.    close();
6. }
7. boolean function open() {
8.    status1 = HT1.initHashTable();
9.    status2 = HT2.initHashTable();
10.   return(status1 AND status2);
11.}

12.procedure next()
13. Input: Operator current from which input tuple t
14. was pushed from input1 or input2
15. do {
16.    if (current is input1) then
17.       i= 1, j = 2
18.    else
19.       i = 2, j = 1
20.    Insert t into hash table HTi
21.    Probe HTj with join keys of t
22.    if(key of HTk,(k=1,2) with key of t is a valid join combinations)
23.       push(t)
24. } while (input1.hasNext() or input2.hasNext())
25.close() {
26.   HT1.close();
27.   HT2.close();
28.}
```

Figura 52 – Algoritmo *pipelining hash-join*
Fonte: adaptado de OZSU; VALDURIEZ, (1999)

Assim, propôs-se, neste trabalho, uma solução baseada no algoritmo *bind-join* denominado *set-bind-join*, cujo principal objetivo é minimizar o volume de dados transferidos ao mediador. Detalhes sobre este algoritmo são apresentados na próxima seção.

8.4.2 *Set-bind-join*

Propõe-se neste trabalho um algoritmo que modela uma variação do algoritmo *bind-join* (HAAS et al, 1997) para o contexto de integração de dados no padrão de *Linked Data*. Diferentemente das operações de junção tradicional, o *set-bind-join* (SBJ) é assimétrico. Essencialmente, o resultado de uma das subconsultas projeta tuplas que incluem os *binds* necessários na outra consulta que participa da junção.

Diferente do DARQ (QUILITZ; LESER, 2008), que executa o *bind-join* com o modelo *tupla-a-tupla,* aumenta a sobrecarga de comunicação entre o *SPARQL endpoint* e o mediador. O algoritmo *set-bind-join*, proposto aqui, processa seguindo a estratégia em que os

dados (tuplas) resultantes da subconsulta que recupera os *binds* são lidos e os valores correspondentes aos *binds* (atributos obrigatórios) são enviados *set-a-set* encapsulados na cláusula SPARQL *BINDINGS* (BUIL-ARANDA; ARENAS e CORCHO, 2011) da outra subconsulta que participa da junção. Essa estratégia é comparável a uma consulta pré-processada, semelhante ao *PreparedStatment* da API JDBC (*Java Database Connectivity*). Um diferencial do SBJ é que explora o paralelismo *intraoperador* executando e transmitindo em paralelo os resultados das subconsultas que incluem os *binds*. Sendo que o resultado de cada subconsulta é *push* (empurrado) para o próximo operador no plano de execução.

A Figura 55 ilustra a execução deste algoritmo, em que se identificam as três principais etapas do algoritmo *set-bind-join.*

a) **Scan** – as fontes de dados são acessadas uma única vez para obter todos os dados com eliminação de duplicatas, juntamente com as propriedades de junção (*binds*) necessárias na próxima fase. A Figura 55(a) ilustra esta etapa.

b) **Set-binds** – transmite/executa vários *bind-joins* em paralelo para diminuir o tamanho dos resultados intermediários e acelerar a avaliação da consulta na etapa posterior. Nesta etapa se explora o parelismo intraoperador, Na Figura 55(b1, c1), observa-se que a cláusula SPARQL *BINDINGS* é usada para filtrar apenas as tuplas que vão compor o resultado final; em seguida os dados são transmitidos ao mediador, de maneira paralela e assíncrona [Figura 55 (b2, c2)].

c) **Junção no mediador** – define a execução da junção final no mediador (*inner-join*) dos dois conjuntos de tuplas, vide Figura 55 (b3, c3), obtidas das fontes de dados. Os resultados de cada subconsulta são *push* imediatamente para o operador subsequente (consumidor) usando o modelo de notificação de eventos. A diferença para o PHJ é que as tuplas são enviadas ao próximo operador em blocos ao invés de tupla-a-tupla.

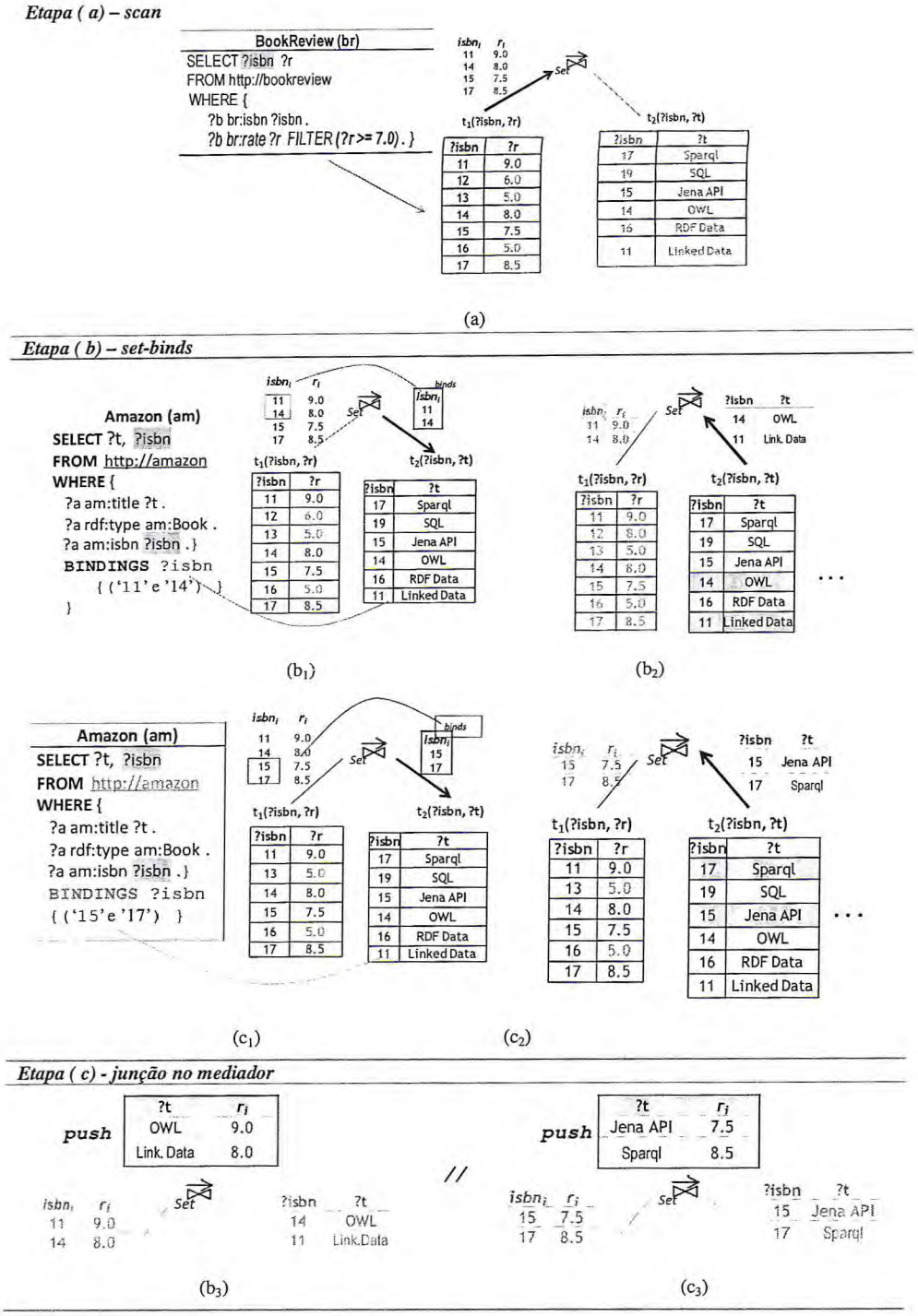


Figura 53 – Execução do algoritmo *set-bind-join*

8.4.2.1 Algoritmo *Set-bind-join*

Nesta seção descreve-se o algoritmo de junção SBJ (Figura 54). O algoritimo recebe duas subconsultas Q_{binds} (a consulta que recupera os *binds*) e Q_2 (a consulta que utiliza os *binds*). Semelhantemente ao algoritmo PHJ, duas tabelas hash (R_{Qbinds}, R_{Q2}) são criadas para armazenar o resultado das duas subconsultas. Os predicados de junção (*binds*) compõem a chave do *hash,* e os outros atributos são os valores associados a cada chave. A variável *n* armazena o total de *binds* recuperados da consulta Q_{binds} e *t* a quantidade máxima de *binds* (*set-binds*) que será enviada em cada subconsulta (este valor é previamente fornecido).

Durante a etapa de *Scan* (linhas 7-14) a fonte de dados F_i é acessada uma única vez pela consulta Q_{binds}. Isto é, se há as seguintes junções a ser executadas na próxima fase $(F_{j1} \bowtie F_i)$, $(F_{j2} \bowtie F_i)$,...,$(F_{jk} \bowtie F_i)$, então é condição suficiente F_i ser acessada uma única vez para gerar $\prod_{Fj1} Fi$, ..., $\prod_{Fjk} Fi$, onde $F_{jm,(1 \leq m \leq k)}$ é a junção de atributos entre F_{jm} e F_i. Na linha 11, todas as tuplas recuperadas de Q_{binds} são armazenados em HT_1 , e na linha 14 obtem-se uma lista com todos os *binds* (chave de HT_1) que serão utilizados na próxima etapa.

As linhas 15-21 refletem a segunda etapa do algoritmo, em que se submetem *n/t* subconsultas Q_2 em paralelo ($Q_{2,binds[1..t]},...,Q_{2,binds[k*t+1...n]}$), ao SPARQL *endpoint* correspondente, sendo $n \geq (t*k+1)$. Observe-se que cada uma dessas subconsultas é executada em uma *thread* t_j (linha 18) independente, e as tuplas obtidas de cada subconsulta são transmitidas ao mediador, de maneira paralela e assíncrona.

Por fim, tem-se a última etapa do algoritmo (linhas 22-28), na qual se faz necessário somente se a subconsulta que recupera os *binds* contribuir com outros valores para resposta da consulta, como ilustrado na Figura 53. Caso afirmativo (linha 24) computa-se para cada thread T_j uma parte da junção (*inner-join*) no mediador das tuplas R_{Qbinds} e R_{Q2} obtidas das subconsultas Q_{binds} e Q_2 respectivamente. O predicado de junção corresponde aos *binds* presentes nos resultados das duas consultas, Assim, para cada par de atributos comuns $(R_{Qbinds}:bind_i ; R_{Q2}:bind_i)$, onde para os valores iguais $R_{Qbinds}:bind_i = R_{Q2}:bind_i$ é feita a junção para compor o resultado final, eliminando atributos redundantes.

Por fim, os resultados de cada *thread* T_j são *push* (linha 29) imediatamente para o operador subsequente (consumidor) usando o modelo de notificação de eventos.

```
Algoritmo: SBJ
1.  Input: SPARQL Queries Q_Binds, Q_2;
2.   Hash Table R_Qbinds, R_Q2 ;
3.   List bind; // bind list
4.   n ; // total of binds retrieved
5.   t ; // maximum of set-binds
     // send for each query
6.  // Scan Phase
7.  Step_1:
8.   R_Qbinds.initHashTable();
9.   //performs bind projections with
     duplicate elimination
10.  R_Qbinds = scan(Q_binds)
11.  // Bind processing phase
12.  //get bind list
13.  binds = R_Qbinds.keys();
14.  n = size(binds)
15. Step_2:
16.  j=1;
17.  for (i=0; i < n; i= i+t)
18.    if(i+t < n)
         launch Thread T_j: submit Q_2
                  with binds[i, i+t]
19.    else
         launch Thread T_j: submit Q_2
                  with binds[i, n-1]
20.    j= j+1;
21.  end for
```

```
Thread T_j:
22.while(tuples available on T_j's output)
23.   read tuples R_Q2 from T_j's output
24.  if R_Qbinds retrieved other values
25.     // evaluate join-at-mediator
26.     Step_3
27.       perform the joins
            (R_Qbinds:bind_i = R_Q2:bind_i)of
          the outputs of R_Qbinds in R_Q2
28.   end if
29. push(R_Q2)
```

Figura 54 – Algoritmo *Set-bind-join*

Este algoritmo usa o paralelismo intra-operador para aproveitar o processamento escalável usando *threads* e a capacidade de paralelização disponível nos processadores *multi-core* modernos. O paralelismo intraoperador é utilizado para reduzir o tempo de execução do algoritmo SBJ e uma questão muito importante em sua utilização é a definição do número de *threads* a serem alocadas. A escolha imprecisa do grau de paralelismo a ser utilizado pode introduzir outros problemas na execução. Por exemplo, a criação de uma grande quantidade de *threads* pode levar a disputa de dados pelo(s) processador(es). Já o contrário poderia aumentar o tempo de execução, pois aumenta o número de *binds* enviado em cada subconsulta.

8.5 Estratégia adaptativa

Um fator importante na integração de dados é a utilização de estratégias de execução adaptativa. Em termos de adaptação, essas estratégias alteram o plano em tempo de execução, re-otimiza a consulta ou utiliza algoritmos específicos para lidar, de forma mais flexível, com acontecimentos imprevisíveis.

Todas as pesquisas mencionadas no Capítulo 4 focam no modelo de custo clássico para integração de dados a fim de melhor estimar o custo de execução das subconsultas. O esforço é para captar previamente informações de fontes de dados, com a intenção de construir o plano de execução ideal. Uma possível solução é a noção de capacidades (*capabilities*) proposta no DARQ (QUILITZ; LESER, 2008). Com base nessas capacidades, o mediador é capaz de determinar as fontes de dados relevantes para responder à consulta do usuário e otimizar as operações de junções com base na seletividade. Enquanto o projeto DARQ usa informações de seletividade fornecidas previamente. Langegger (2010) apresentou uma abordagem que se baseia em estatísticas RDF mais representativas, incluindo histogramas. As estatísticas são geradas automaticamente e podem ser solicitadas por meio de uma extensão para o protocolo SPARQL, porém o inconveniente dessa solução é que para cada SPARQL *endpoint* se deve associar um coletor de estatísticas.

De fato, tendo informações estatísticas de qualidade, a definição de um plano de execução otimizado torna-se muito mais precisa. Todavia, mesmo o plano ideal enfrenta o problema de se tornar ineficiente em tempo de execução devido: a) à ocorrência de eventos imprevisíveis durante a execução das subconsultas; b) às variações no tempo de resposta dos SPARQL *endpoints*; e c) a falta de previsão do volume de dados retornado por uma consulta pode comprometer o desempenho do processamento das subconsultas.

A determinação do melhor plano nesse cenário é uma tarefa difícil, e estratégias puramente estáticas não são capazes de modelar tais variações. Já uma estratégia adaptativa pode capturar eventos imprevisíveis que podem ocorrer durante a execução de uma consulta recorrendo aos algoritmos que sejam mais apropriados a cada instante.

Tomando-se uma análise isolada em cada algoritmo de junção apresentado na seção 8.4, nem sempre o emprego de um único algoritmo de junção implica bom desempenho global. Em termos gerais, não existe um algoritmo que apresente o melhor desempenho em todos os casos, e deve-se aproveitar, quando possível, do processamento paralelo disponível e/ou da redução de resultados intermediários para obter planos de execução o mais próximo possível do ótimo.

Por exemplo, duas consultas de Q_i e Q_j podem ser executadas em paralelo. Porém, utilizar sempre essa estratégia pode não ser ideal, uma vez que o resultado prévio (*binds*) da consulta Q_i pode ajudar sobremaneira a reduzir a quantidade de dados transferidos ao mediador por uma consulta Q_j.

Nesse sentido, houve outro direcionamento nesta pesquisa em relação a trabalhos relacionados (QUILITZ; LESER, 2008; LANGEGGER, 2010). Neste trabalho foi proposta uma estratégia adaptativa que possibilita mudar em tempo de execução o algoritmo de junção recorrendo aos algoritmos mais adequados a cada instante, aproveitando os dados já recuperados.

A seguir apresenta-se a geração do plano inicial e a estratégia adaptativa proposta.

8.5.1 Geração do plano de execução inicial

O objetivo é fornecer uma estratégia de fácil cômputo e não onerosa para gerar um plano de execução inicial que servirá de entrada para a estratégia adaptativa que monitora a execução dos algoritmos de junção, de modo que na ausência de estatísticas sobre as fontes de dados, define-se uma heurística bem simples que inicialmente prioriza a diminuição do volume de dados intermediários provenientes dos SPARQ *endpoints* com algoritmo SBJ; e as junções subsequentes são executadas no mediador por meio do algoritmo PHJ.

Por exemplo, considere-se o plano gerado na etapa de reformulação ilustrado no Capítulo 6. Para a geração do plano de execução inicial utiliza-se informação das fontes primárias e secundárias, identificadas pelo algoritmo de reformulação de consultas, como critério para definir o algoritmo de junção de forma que se utiliza o algoritmo SBJ. As consultas sobre as fontes secundárias recuperam os *binds* que servirão de entrada para a consulta sobre a as fontes primárias. Observe na Figura 57 uma seta de *Q3'* para *Q2"* que indica que os *binds* de *Q3'* serão utilizados por *Q2"*.

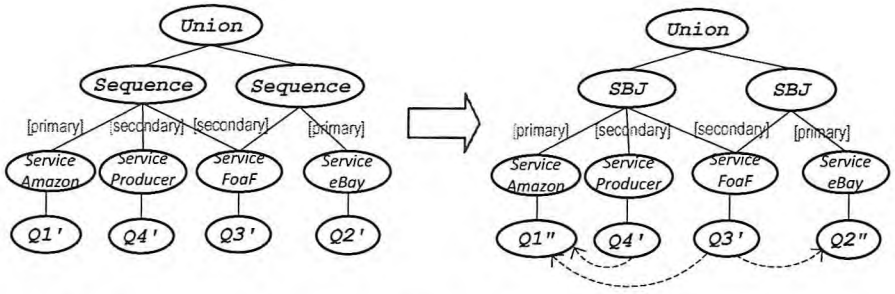


Figura 55 – Geração do Plano otimizado inicial na ausência de estatísticas

8.5.2 Processamento adaptativo dos algoritmos de junção

A estratégia adaptativa deve ser utilizada levando em consideração minimizar a transmissão de resultados intermediários ao mediador, o tempo total de execução, e o tempo de resposta inicial. Para tanto, faz-se uma sintonia fina entre os objetivos de aumento do *througput*/paralelização do processamento de subconsultas (algoritmo PHJ) e a redução do volume de dados transmitidos (algoritmo SBJ), mudando o algoritmo dinamicamente durante a execução sem interromper o envio de dados do SPARQL *endpoint* para o mediador. Para permitir isso, utilizou-se o padrão *Strategy* (GAMMA et al, 1995), que encapsula cada algoritmo e os faz intercambiáveis utilizando o princípio do polimorfismo. Isso permite mudar o algoritmo sem prejudicar inadvertidamente o processamento já realizado.

Dessa forma, monitoram-se dois parâmetros durante a execução: atrasos na chegada dos dados provenientes dos SPARQL *endpoints* (*TEMPO_LIMITE*) e a quantidade de dados que chegam ao mediador (*TAM_MAX*).

As Figuras 58 e 59 exibem um fragmento do algoritmo que implementa a estratégia adaptativa com uma sintaxe estilo *java*. Explora-se uma recente API de programação concorrente (*java.util.concurrent*). O uso dessa API ajuda a diminuir a complexidade no processamento paralelo, assim como tirar maior proveito dos recursos dos processadores modernos de múltiplos núcleos.

A adaptação a partir do PHJ é apresentada na Figura 56. Nas linhas 5 e 13 verifica-se qual consulta terminou primeiro (*Q1* ou *Q2*). Por exemplo, caso *Q1* tenha concluído primeiro, a estratégia adaptativa será acionada somente se duas condições forem satisfeitas: a) se *Q1* obter uma quantidade de tuplas inferior ao tamanho máximo (*TAM_MAX*) (linha 7); e b) se *Q2* ainda não realizou entrega de dados (linha 7). Comportamento similar ocorre se a consulta *Q2* terminar antes de *Q1* (linhas 13-19).

```
1. Future<Collection<String>> Q1 = monitor_thread.submit(query1);
2.  Future<Collection<String>> Q2 = monitor_thread.submit(query2);
3.  // adapted from the PHJ
4.  while (!monitor_thread.isTerminated()) {
5.     if ( Q1.isDone() ) {
6.       int n = Q1.get().size();
7.       if (n < TAM_MAX ) || (!Q2.getTuples()) {
8.          Q2.stop();
9.          Q2.setListaBinds(Q1.get());
10.         Q2 = monitor_thread.submit(query2);
11.         break;
12.   } }
13.   if ( Q2.isDone() ) {
14.     int n = Q2.get().size();
15.     if (n < TAM_MAX )|| (!Q1.getTuples()) {
16.        Q1.stop();
17.        Q1.setListaBinds(Q2.get());
18.        Q1 = monitor_thread.submit(query1);
19.        break;
20.   } }
21. }
22. ...
```

Figura 56 – Fragmento do algoritmo da estratégia adaptativa a partir do algoritmo PHJ

A adaptação a partir do SBJ (Figura 57) acontece se a consulta que recupera os *binds* (*n*) ultrapassar o tamanho máximo (linha 6), neste caso, sai do laço e muda para o algoritmo PHJ (linha 9). Na linha 13 inicia-se a execução de *Q2* com algoritmo PHJ. Senão, a lista com os *binds* fora anexada (linha 7) e, continua-se com algoritmo SBJ.

Quanto menor for o valor de *n* menos complexas serão as subconsultas que incorporam os *binds*. No Capítulo 9, apresenta-se uma análise empírica da variação do tamanho dos *binds* que ajudam a definir o seu tamanho ideal.

```
1. Future<Collection<String>> bindQuery = monitor_thread.submit(bindQuery);
2.  // adapted from the SBJ
3.  while (!monitor_thread.isTerminated()) {
4.    if (bindQuery.isDone()) {
5.      n = bindQuery.get().size();
6.      if (n < TAM_MAX ) {
7.         Q2.setListaBinds(bindQuery.get());
8.      }
9.      break;
10.  }
11. }
12. if (q2.getListaBinds() == null)
13.     Future<...> fQuery2 = phj(Q2);
14. else
15.     Future<...> fQuery2 = sbj(Q2);
16. ...
```

Figura 57 – Fragmento do algoritmo da estratégia adaptativa a partir do algoritmo SBJ

8.6 Considerações do capítulo

Neste capítulo foram analisadas técnicas conhecidas envolvendo operações de junção que exploram a natureza dos *Linked Data*, destacando-se redução do volume de dados intermediários, processamento paralelo do plano de execução. Foi apresentada a estratégia adaptativa que possibilita mudar em tempo de execução o algoritmo de junção recorrendo aos algoritmos mais adequados a cada instante.

Para avaliar a consulta de acordo com o plano, executa-se cada operador em uma *thread* separada. A comunicação entre os operadores é baseada em uma estrutura de *hash* compartilhada para permitir o processamento de consulta paralela.

No Capítulo 9 apresenta-se uma avaliação da escalabilidade da solução proposta, valida-se empiricamente os algoritmos de junção e a estratégia de execução.

9 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta diversas análises de resultados de experimentos. Para validar a estratégia de execução, foram implementados os algoritmos de junção e a estratégia adaptativa descritos no Capítulo 8. O objetivo dos experimentos é avaliar, para um escopo limitado de consultas, porém representativas, a adequação do método de processamento das consultas SPARQL descrito no Capítulo 5, em especial as etapas de otimização global e a execução e composição do resultado. Além disso, foram analisados aspectos técnicos sobre os algoritmos de junção em ambiente distribuído em comparação com tempo de resposta de consultas federadas *ARQ/Service*. A última seção apresenta um sumário dos resultados e a conclusão do Capítulo.

9.1 Detalhes de implementação

Foi desenvolvido um processador de consultas SPARQL federadas em Java utilizando o *framework* Jena/ARQ. O processador contempla a implementação dos algoritmos de junção, estratégia de execução adaptativa. Utilizou-se o padrão de projeto *Observer* para notificação de eventos entre o produtor e consumidor dos dados e adotou-se o padrão de projeto *Strategy* para intercambiar os algoritmos em tempo de execução.

O *framework* Jena é um projeto de código aberto desenvolvido originalmente pela HP (*Hewlett-Packard*) que possui um conjunto de funcionalidades para apoiar o desenvolvimento de aplicações no contexto da Web Semântica. Além das funcionalidades para manipulação da linguagem OWL, o Jena possui recursos para gerenciar linguagens como RDF, RDFS (CARROLL et al, 2004). Outro recurso interessante é a funcionalidade para uso da linguagem SPARQL de forma programática.

No *framework* Jena/ARQ, os arquivos de instâncias das fontes de dados necessitam ser carregados em memória sempre que o *engine* processar as consultas. Entretanto, alguns experimentos consultam fontes de dados com milhões de triplas, o que torna impraticável carregar todas as informações em memória, devido às limitações de espaço de memória dos computadores utilizados nos experimentos.

Diante dessa limitação, tornou-se necessário um mecanismo de armazenamento mais eficiente para que seja possível o tratamento de grandes volumes de dados. Essa

limitação de desempenho pode ser tratada utilizando banco de dados que possuem mecanismos para otimizar essas tarefas. Na presente implementação, optou-se pela utilização do Jena *Tuple DataBase* (JENA/TDB, 2010), que é um módulo interna do *framework* Jena e se constitui de um sistema de armazenamento em banco de dados relacionais especializado para dados no padrão de *Linked Data*.

9.1.1 Preparação dos experimentos

Para a execução de experimentos, é necessário, previamente, definir os objetivos; planejar a execução, como o ambiente onde serão executados com as configurações; e definir uma metodologia de execução que será utilizada para permitir uma análise conclusiva dos resultados.

9.1.2 Objetivos

Os experimentos estão divididos em três grupos. Para o primeiro grupo, é analisado o efeito da variação no tamanho dos *binds* no algoritmo SBJ, a análise desses resultados auxilia na parametrização dos experimentos que seguem. O segundo grupo compara o tempo de resposta das consultas federadas usando o processador *ARQ/Service* (ARENAS; PEREZ, 2011) com os algoritmos PHJ, SBJ e a estratégia adaptativa. A análise dos resultados deste grupo de experimentos contribui principalmente para demonstrar o ganho de desempenho da estratégia proposta em relação ao processador de consultas federadas *ARQ/Service*. O terceiro grupo de experimentos foi projetado com o objetivo de investigar a escalabilidade dos algoritmos de junção juntamente com a validação da estratégia adaptativa, que acessam fontes de dados com tamanhos variados.

Esses cenários norteiam todo o planejamento dos experimentos, como a preparação do ambiente (visto na próxima seção) e a elaboração da metodologia de execução apresentada na seção 9.2.3.

9.1.3 Ambiente de execução

O processador de consultas SPARQL foi implantado num computador separado (com processador *Pentium Dual Core de 1,8 GHz com 2 GB* de memória RAM e sistema operacional *Windows 7*, endereço IP*: 192.168.100.24, Java Runtime Environment 6, 32 bits*, e o *Jena/ARQ*).

Há dois cenários para o ambiente de execução:

a) consultas submetidas a SPARQL *endpoints* reais disponíveis na Web;

b) consultas submetidas aos SPARQL *endpoints* implantados com o Jena/TDB e configurados numa rede local. Para este ambiente, os SPARQL *endpoints* foram instalados em quatro computadores, sendo que cada computador foi configurado com o MySQL e o Jena/TDB. A Figura 58 ilustra a distribuição dos SPARQL *endpoints* nos computadores do ambiente em que estão conectados por meio de uma rede local *ethernet* de 10Mbits. Em seguida, apresenta-se uma descrição da configuração de cada computador.

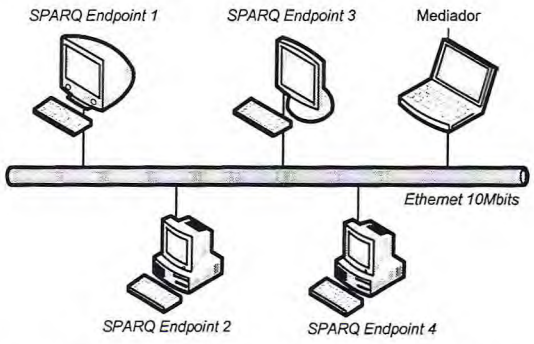


Figura 58 – Ambiente para execução dos experimentos no cenário 2

SPARQL Endpoint 1: processador AMD Athlon^{TM} X2 de 2.8 GHz com 2 GB de memória RAM e sistema operacional windows Vista Business, endereço IP: 192.168.100.41.

SPARQL Endpoint 2: processador AMD Athlon™ X2 de 2.8 GHz com 2 GB de memória RAM e sistema operacional Windows Server 2008, endereço IP: 192.168.100.21.

SPARQL Endpoint 3: processador AMD Athlon™ X2 de 2.8 GHz com 2 GB de memória RAM e sistema operacional Linux Ubuntu Server, endereço IP: 192.168.100.22.

SPARQL Endpoint 4: processador Pentium Dual™ Core de 1,8 GHz com 2 GB de memória RAM e sistema operacional Windows 7, endereço IP: 192.168.100.23.

9.1.4 Metodologia de execução dos experimentos

Para que os resultados dos experimentos sejam relevantes aos objetivos do trabalho, é necessário elaborar um planejamento das atividades que contemplem todos os resultados desejados.

A metodologia de execução dos experimentos consiste na obtenção do tempo médio de dez execuções da mesma consulta (plano de execução), desprezando a primeira execução ("tempo frio"). Cada ambiente de execução possui um propósito, o que possibilita a comparação dos resultados de forma a permitir fazer uma avaliação do desempenho das consultas em diferentes configurações. Por fim, as consultas são executadas automaticamente por uma aplicação cliente desenvolvidas para este propósito.

9.1.5 Fontes de dados

Nesta seção, apresentam-se as fontes de dados utilizadas nos experimentos para dois cenários, como segue.

Cenário 1 – utilizam-se fontes de dados que abrange vários domínios de interesse popular, incluindo notícias, filmes música, farmacêutico e medicamentos.

A Figura 59 relaciona as fontes de dados do cenário 1, destaca a abrangência e apresenta uma breve descrição de cada fonte.

Fontes de dados/SPARQL *endpoint*	Cobertura	Descrição
DBPedia (http://dbpedia.org/sparql)	Genérico (temas variados)	É um esforço comunitário para extrair informações estruturadas da Wikipedia.
LinkedMDB (http://data.linkedmdb.org/sparql)	Filmes, atores, música	Um banco de dados de filmes, atores e músicas.
DrugBank (http://www4.wiwiss.fu-berlin.de/diseasome/sparql)	Drogas, composição e interações	Abrange compostos químicos, reações e drogas, composição e interações com outras drogas.
Dailymed (http://www4.wiwiss.fu-berlin.de/dailymed/sparql)	Drogas comercializadas	Fornece informações sobre a grande parte dos medicamentos comercializados, incluindo informações gerais sobre a estrutura química do composto e finalidade terapêutica, detalhes sobre a farmacologia clínica do composto, indicação e uso, contraindicações, advertências, precauções, reações adversas, superdosagem e aconselhamento de pacientes.

Figura 59 – Fontes de dados utilizadas nas consultas do cenário 1

Cenário 2 – as fontes de dados possuem o mesmo vocabulário dos esquemas-fonte descritos no Capítulo 5. Recapitulando, há duas fontes que descrevem os dados sobre as lojas virtuais *Amazon* e *eBay*, uma terceira fonte de dados que possui informações detalhadas sobre os fabricantes de produtos (*Producer*) e uma quarta fonte de dados com informações dos consumidores que postaram revisões (*Foaf*).

Os arquivos RDF que representam as fontes de dados foram coletados do site *The Berlin SPARQL Benchmark* (BIZER; SCHULTZ, 2008) e estão livremente disponíveis para *download* na Internet[15]. Uma grande motivação para a utilização dessas fontes de dados é que estão disponíveis em diferentes tamanhos, o que permite uma avaliação mais abrangente da estratégia de otimização.

As Tabelas 1, 2, 3 e 4 apresentam cada fonte de dados com diversos tamanhos, de forma a obter resultados capazes de ser analisados de acordo com os objetivos definidos.

[15] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/V1/results/index.html
http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/index.html

Tabela 1 – Fontes de dados *Amazon*

Nº Classes	Fonte de Dados ***Amazon***		
	F_{A1}	F_{A2}	F_{A3}
Produtos	1.915	9.609	48.172
Ofertas	*55.700*	*192.180*	*1.416.240*
Revisões	27.850	240.225	708.120
Formato do arquivo original	*Turtle*	*N-Triple*	*Turtle*

Tabela 2 – Fontes de dados *eBay*

Nº Classes	Fonte de Dados ***eBay***	
	F_{E1}	F_{E2}
Produtos	477	1.915
Tipos de produtos	55	151
Revisões	6.660	27.850
Fabricantes	10	60
Formato do arquivo original	*N-Triple*	*N-Triple*

Tabela 3 – Fontes de dados *Producer*

Nº Classes	Fonte de Dados – ***Producer***		
	F_{P1}	F_{P2}	F_{P3}
Fabricantes	60	199	1422
Formato do arquivo original	*Turtle*	*N-Triple*	*Turtle*

Tabela 4 – Fontes de dados *FoaF*

Nº Classes	Fonte de Dados ***FoaF***		
	F_{F1}	F_{F2}	F_{F3}
Revisores	339	1432	12.351
Formato do arquivo original	*N-Triple*	*Turtle*	*N-Triple*

9.1.6 Consultas SPARQL utilizadas

Cenário 1 – a Figura 62 mostra três consultas federadas sobre as fontes de dados *DrugBank*, *Dailymed* e *DBPedia*, *LinkedMDB*, respectivamente.

$Q_{1.1}$: recupera o nome completo das drogas para o tratamento da doença de código 3149

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/.../>
PREFIX dailymed: <http://www4.wiwiss.fu-
berlin.de/dailymed/.../dailymed/>
SELECT DISTINCT ?dg ?dgn
WHERE {
  SERVICE <http://www4.wiwiss.fu-berlin.de/diseasome/sparql> {
     <http://www4.wiwiss.fu-berlin.de/diseasome/resource/diseases/3149>
                                          diseasome:possibleDrug ?dg .
 }

 SERVICE <http://www4.wiwiss.fu-berlin.de/dailymed/sparql> {
    ?dg dailymed:fullName ?dgn .
 }
}
```

$Q_{1.2}$: recupera o nome completo de todas as drogas (?dgn) registradas no *DrugBank* (?dg)

```
PREFIX diseasome: <http://www4.wiwiss.fu-berlin.de/diseasome/.../>
PREFIX dailymed: <http://www4.wiwiss.fu-
berlin.de/dailymed/.../dailymed/>
SELECT DISTINCT ?dg ?dgn
WHERE {
  SERVICE <http://www4.wiwiss.fu-berlin.de/diseasome/sparql> {
     ?d diseasome:possibleDrug ?dg .
  }

  SERVICE <http://www4.wiwiss.fu-berlin.de/dailymed/sparql> {
    ?dg dailymed:fullName ?dgn .
  }
}
```

$Q_{1.3}$: recupera o nome e a data de aniversário dos atores que participaram do filme *Lassie Come Home* (38394)

```
PREFIX movie: <http://data.linkedmdb.org/resource/movie/>
PREFIX dbpedia: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?actor_name ?birth_date WHERE {
 SERVICE <http://data.linkedmdb.org/sparql> {
    <http://data.linkedmdb.org/resource/film/38394> movie:actor ?actor .
    ?actor movie:actor_name ?actor_name .
  }

 SERVICE <http://dbpedia.org/sparql> {
    ?actor2 a dbpedia:Actor .
    ?actor2 foaf:name ?actor_name_en .
    ?actor2 dbpedia:birthDate ?birth_date .
    FILTER(STR(?actor_name_en) = ?actor_name)
 }
}
```

Figura 60 – Consultas federadas sobre as fontes de dados do Cenário 1

Cenário 2 – considerem-se duas consultas SPARQL sobre a ontologia de domínio *Sales* (Figura 26) apresentadas na Figura 61, juntamente com uma breve descrição.

$Q_{2.1}$: recupera o título (*?t*) dos produtos fabricados nos Estados Unidos

```
PREFIX s:<http://sales/>
PREFIX rdf:<...>
SELECT ?t
WHERE {
  ?p rdf:type s:Product .
  ?p s:title ?t .
  ?p s:producer ?pdr .
  ?pdr s:pCountry ?cy . FILTER (?cy, 'USA')
}
```

$Q_{2.2}$: recupera os títulos (*?t*) dos produtos fabricados nos Estados Unidos e avaliados por consumidores da França, cuja *nota* é maior do que oito (*?rt* > *8*)

```
PREFIX s:<http://sales/>
PREFIX rdf:<...>
SELECT ?t ?cm ?r
WHERE {
  ?r rdf:type s:Review .
  ?r s:comment ?cm .
  ?r s:rate ?rt . FILTER (?rt > 8 )
  ?r s:hasReviewer ?rvr .
  ?rvr s:country ?ct . FILTER(?ct, 'FR')
  ?r s:reviewFor ?p .
  ?p s:title ?t .
  ?p s:producer ?pdr .
  ?pdr s:pCountry ?cy FILTER(?cy, 'USA')
}
```

Figura 61 – Consultas federadas sobre as fontes de dados do Cenário 2

9.2 Análise dos resultados

Os testes foram realizados com tomadas de tempos de resposta das consultas distribuídas. Para cada plano de execução foram feitas dez execuções, sendo que a primeira execução é descartada (o "tempo frio") e considerado o tempo médio das demais tentativas (o "tempo quente").

As medidas monitoradas durante a execução são o tempo de resposta das consultas distribuídas. A comparação dos tempos médios de execução das consultas nos diferentes cenários foi feita por meio de gráficos que apresentam os tempos de execução das consultas de forma normalizada, conforme discutido a seguir.

9.2.1 Análise do algoritmo SBJ variando o tamanho dos *binds*

Neste cenário avaliou-se o efeito da variação no parâmetro t (tamanho dos *binds* do algoritmo SBJ) com o objetivo de auxiliar em sua definição para os próximos experimentos. A primeira avaliação é desenvolvida no contexto das fontes de dados em uma rede local, em que se tem controle maior do tamanho das fontes de dados, seguido de uma análise no ambiente Web, bem mais imprevisível e de maior variabilidade nos tempos de resposta.

Executou-se a consulta $Q_{2.1}$, que recupera apenas os títulos do produtos fabricados nos Estados Unidos. Para esta consulta foram utilizadas fontes de dados com variabilidades diversas em relação ao tamanho para observar de maneira mais abrangente o comportamento do algoritmo SBJ.

As fontes utilizadas foram *Amazon* (F_{A1}, F_{A2}) e *Producer* (F_{P1}, F_{P2}, F_{P3}), que geraram uma combinação de seis gráficos em que o eixo y corresponde ao tempo de resposta da consulta em milissegundos e o eixo x exibe o tamanho dos *binds* enviados em cada subconsulta sobre a fonte de dados *Amazon*. Para os gráficos seguintes o tempo de resposta será medido em segundos.

Para cada fonte de dados, optou-se por uma análise extensiva que variou de *1*, *2*, ..., n, onde n representa o total de *binds* recuperados na consulta sobre a fonte de dados *Producer*. Os resultados desses experimentos são apresentados nas Figuras 64 e 65 e analisados a seguir.

De maneira geral, quando $t \geq n/2$ os ganhos no tempo resposta são maiores. Deve ser observado também que para $n/2 \leq t \leq n$ a variabilidade no tempo de resposta é pequena conforme ilustrado nos gráficos das Figuras 64 e 65. Fazendo-se uma análise mais minuciosa na Figura 64 nos gráficos (a) e (b), os resultados foram sensivelmente melhores para $t = n$; já os gráficos (c) e (d) apresentaram resultados sensivelmente melhores para $n/2 < t < n$; enquanto nos gráficos (e) e (f), em que a quantidade de *binds* é consideravelmente maior (um total de 623), os melhores tempos foram para valores de $t \approx n/2$, observando-se pequena piora no tempo de resposta quando o valor de t se aproxima de n. Assim, estes resultados mostram que o parâmetro t não pode crescer indefinidamente.

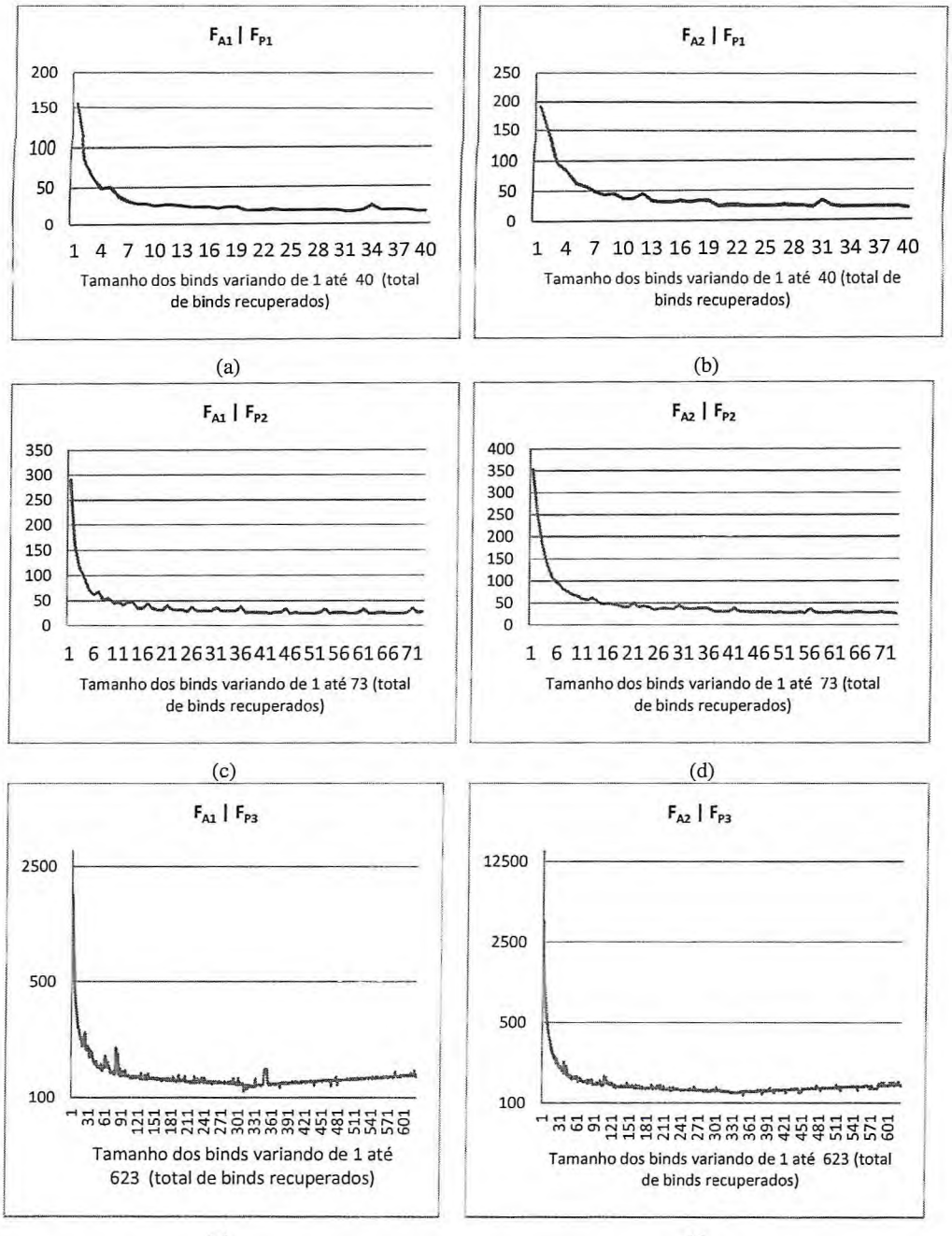
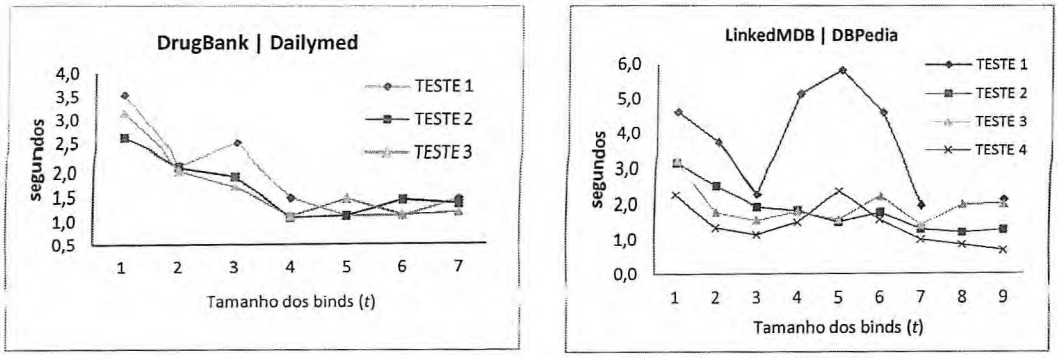


Figura 62 – Resultados do algoritmo *set-bind-join* com variação no tamanho dos *binds* para consulta $Q_{2.1}$

Na Figura 63, apresenta-se uma análise com consultas submetidas a SPARQL *endpoints* reais na Web, os resultados da Figura 63(a) reforçam que a estratégia *bind-join*, que segue o modelo *tupla-a-tupla,* não é adequada, sendo que os melhores tempos são para $t \geq n/2$.



(a) consulta $Q_{1.1}$ (b) consulta $Q_{1.2}$

Figura 63 – Resultados do algoritmo SBJ com variação no tamanho dos *binds* das consultas $Q_{1.1}$ e $Q_{1.2}$

Os resultados ilustrados nas Figuras 64 e 65 apresentam evidências de que o tempo de resposta é pior quando o valor do parâmetro de entrada t é igual a *1*. Esta parametrização ($t=1$) corresponde ao algoritmo *bind-join* que segue um modelo tupla-a-tupla usado no projeto DARQ (QUILITZ; LESER, 2008), ficando claro em todos os gráficos que a perda de desempenho é considerável.

A partir dessas avaliações pode-se definir com mais clareza o parâmetro t. Sendo que, para os próximos experimentos define-se uma regra simples de configuração: se ($n < 100$) então $t \leftarrow n$, caso contrário, $t \leftarrow n/2$.

É importante destacar também que o tempo de resposta de consulta submetida ao mesmo SPARQL *endpoint* na Web (vide Figura 65), de maneira geral, mostrou-se mais instável, com surtos momentâneos no tempo de resposta.

No gráfico da Figura 65(a), as variações no tempo de resposta dos três testes não foram tão significativas e todas as consultas obtiveram uma resposta. Enquanto que no gráfico da Figura 65(b), principalmente no SPARQL *endpoint DBpedia* o efeito da imprevisibilidade foi mais evidente, sendo que, muitas vezes simplesmente não se consegue responder às consultas, nas primeiras tentativas. Dessa forma, mesmo que o plano de execução inicial seja o melhor em um determinado instante, baixas taxas de entrega dos dados ou erros no SPARQL *endpoint* (Erro HTTP 500) no momento da execução da consulta podem tornar o plano ineficiente.

A variabilidade apresentada pela consulta $Q_{1.2}$ não é benéfica para o desempenho do plano, pois o sistema vai experimentar atrasos ou indisponibilidade na entrega de dados em certos instantes. Neste caso, o algoritmo PHJ lida melhor com estes problemas, o que confirma a importância de uma estratégia adaptativa que se adéque, em tempo de execução, a latências não previstas.

9.2.2 Cenário 1 – SPARQL endpoints na Web

Consulta $Q_{1.1}$

Para a primeira consulta realizada sobre os SPARQL *endpoints DrugBank* e *Dailymed*, foi observado na Figura 64 que o pior desempenho ocorreu com a utilização do algoritmo PHJ, pois a consulta sobre o SPARQL *endpoint Dailymed* retorna expressivo volume de dados, considerando que nenhum filtro é aplicado. Já a utilização do algoritmo SBJ trouxe um ganho de desempenho perceptível, em que se recuperou primeiro os resultados das possíveis drogas (*?dg*) da doença de código "3149". O resultado dessa consulta foi usado como *bind* na consulta sobre o *endpoint Dailymed*. Os benefícios do algoritmo SBJ advêm da redução sobremaneira da quantidade de dados intermediários enviado ao mediador. Os tempos de resposta mostraram que o SBJ apresentou tempo 8,7 vezes melhor em comparação com o resultado do processador *ARQ/service* e 12,6 vezes melhor em relação ao PHJ.

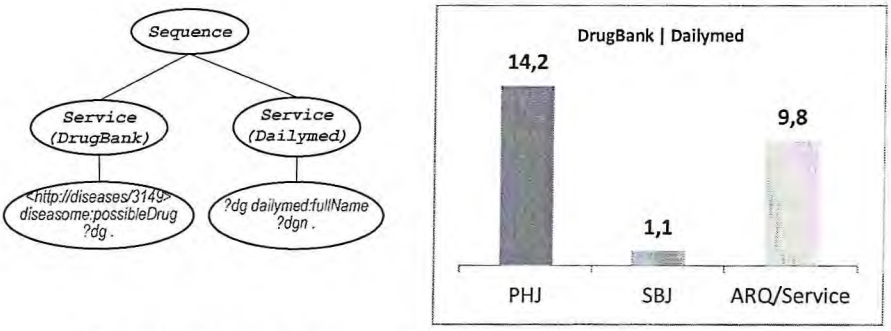


Figura 64 – Gráficos de comparação dos resultados da consulta $Q_{1.1}$

Consulta $Q_{1.2}$

Para a segunda consulta ($Q_{1.2}$) realizada sobre os SPARQL *endpoints DrugBank* e *Dailymed*, foi observado (Figura 65) que o melhor desempenho ocorreu com o algoritmo PHJ, pois a consulta sobre ambos os SPARQL *endpoint* (*Dailymed* e *DrugBank*) retornaram expressivo volume de dados, considerando que nenhum filtro é aplicado. Outra vantagem deste algoritmo é a geração de resultados o mais cedo possível, verificou-se que, na média,

gastaram-se 11,6 segundos para geração do primeiro resultado, de um total de 28,8 segundos. A importância deste resultado é que o mesmo é disponibilizado (*push*) imediatamente para o operador subsequente (consumidor), usando o modelo de notificação de eventos, conforme apresentado no Capítulo 8.

A execução do algoritmo SBJ ocorreu com a seguinte configuração, recuperou-se primeiro o código de todas as drogas (*?dg*) cadastradas no *DrugBank*, que é a consulta que recupera o menor número de *binds*, um total de 936. O resultado dessa consulta foi usado como *bind* na consulta sobre o *endpoint Dailymed*, sendo que, considerou-se o parâmetro t = *936/2 = 468*. Verificou-se um tempo de resposta total de aproximadamente 8% pior comparado com o PHJ, isso ocorreu porque se recuperou um número considerável de *binds*, não conseguindo uma redução significativa da quantidade de resultados intermédiarios transmitidos ao mediador.

Para esta mesma consulta com o processador *ARQ/Service*, em nenhum dos dez experimentos recuperaram-se os resultados. Detectou-se, quando o volume de dados recuperado é relativamente grande, que o processador *ARQ/Service* não consegue obter o resultados. Isto reforça a importância da implementação dos algoritmos propostos para lidar com volume de dados crescente (escalabilidade) e a geração de resultados o mais cedo possível.

Por sua vez, os resultados das consultas $Q_{1.1}$ e $Q_{1.2}$ motivam a necessidade de uma estratégia adaptativa que se beneficie das vantagens dos algoritmos SBJ e PHJ.

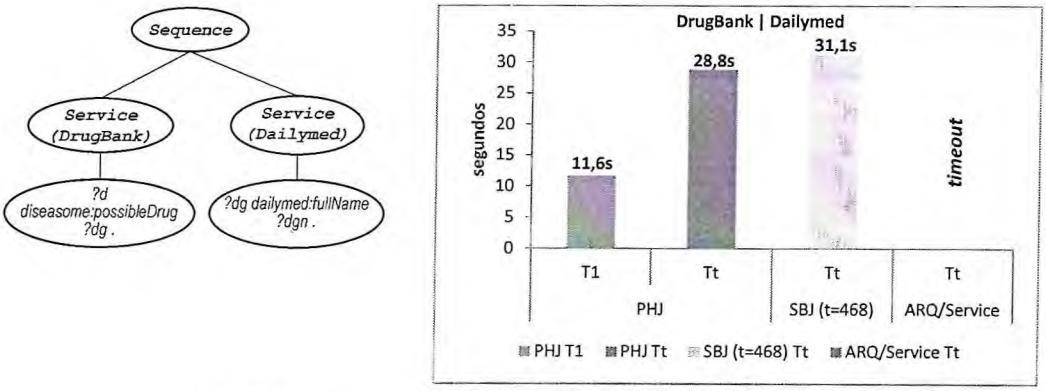


Figura 65 – Gráficos de comparação dos resultados da consulta $Q_{1.2}$

Consulta $Q_{1.3}$

Uma observação importante na análise da consulta $Q_{1.3}$ (Figura 66) é que o algoritmo PHJ não recupera todos os resultados possíveis. Este problema ocorreu devido a uma limitação imposta pelo SPARQL *endpoint DBpedia* que limita a quantidade de dados retornados. Quando se executa a consulta sobre o *DBpedia* com PHJ, nenhuma restrição é imposta, o que gera uma quantidade significativa de resultados intermediários. Este problema não ocorre com o SBJ que filtra previamente o nome dos atores da fonte *LinkedMDB*, o que reduz consideravelmente a quantidade de dados recuperados da consulta sobre *DBpedia*.

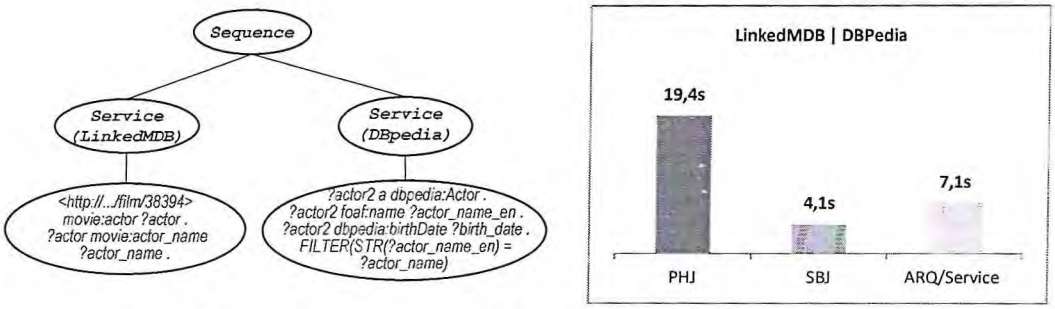


Figura 66 – Gráficos de comparação dos resultados da consulta $Q_{1.3}$

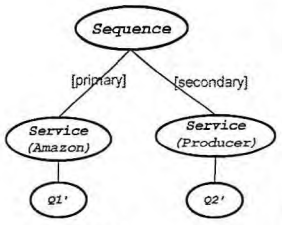
9.2.3 Cenário 2 – Rede Local com Jena/TDB

Foram executadas as duas consultas apresentadas na Figura 61. Para cada consulta, aplicou-se o algoritmo de reformulação de consulta (vide seção 7.3) para geração do plano de execução. Este grupo de experimentos não foi executado com o *ARQ/Service* devido às consultas com Jena/TDB ainda não dar suporte a consultas federadas usando a cláusula *Service*.

Consulta $Q_{2.1}$ – Análise dos algoritmos PHJ e SBJ

O objetivo desta consulta é testar a escalabilidade dos algoritmos PHJ e SBJ variando o tamanho das fontes. Também se analisa o tempo da primeira resposta gerada pelo

algoritmo PHJ. Para atingir este objetivo, considerou-se uma parte do plano que está ilustrado na Figura 67(a) e as subconsultas Q_1' e Q_2' apresentadas na Figura 67(b).



(a)

Q1' (Amazon)

```
SELECT ?t ?pdr
WHERE {
 ?p rdfs:label ?t .
 ?p a:producer ?pdr.
}
```

Q2' (Producer)

```
SELECT ?pdr
WHERE {
  ?pdr p:pCountry ?cy .
  FILTER (?cy, 'USA')
}
```

(b)

Figura 67 – (a) plano de execução da consulta $Q_{2.1}$; (b) subconsultas Q_1' e Q_2'

Os gráficos da Figura 68 apresentam os tempos de resposta dos algoritmos PHJ e SBJ, sendo, também, computada a primeira resposta (T_1) do algoritmo PHJ.

Conforme já mencionado, o algoritmo SBJ busca reduzir a transmissão de resultados intermediários e o algoritmo PHJ se preocupa em explorar ao máximo o paralelismo intraoperador, gerando resultados tão logo estejam disponíveis. O tempo do primeiro resultado (T_1) foi melhor em todos os testes, conforme esperado, e cabe ressaltar que esses resultados podem ser consumidos automaticamente pelo próximo operador no plano de execução, seguindo o modelo *push* de entrega de dados.

Comparando-se os tempos totais (T_t) obtidos pela execução de cada algoritmo, verifica-se que o tempo total (T_t) varia de acordo com o tamanho dos possíveis *binds* obtidos da consulta Q_2'. No caso do gráfico da Figura 68(a), em que os *binds* são menores, o tempo de resposta total é quase 20% menor que o algoritmo PHJ. Para o gráfico da Figura 68(b), em que Q_2' recuperou um número maior de *binds*, o ganho foi menor, próximo de 10%, em comparação com o tempo total do PHJ.

Estes resultados apresentam evidências de que, quanto menor o tamanho dos *binds*, menor é o tempo de resposta. Esta diminuição advém da redução da carga de processamento no mediador, eliminando previamente resultados intermediários que não contribuem para o resultado.

No caso do gráfico da Figura 68(c), o algoritmo PHJ obteve resultados melhores. Isso ocorre porque na execução do algoritmo SBJ se recuperou um número significativamente maior de *binds*. Esses experimentos reforçam a necessidade de uma estratégia adaptativa que se beneficia das vantagens dos algoritmos SBJ e PHJ.

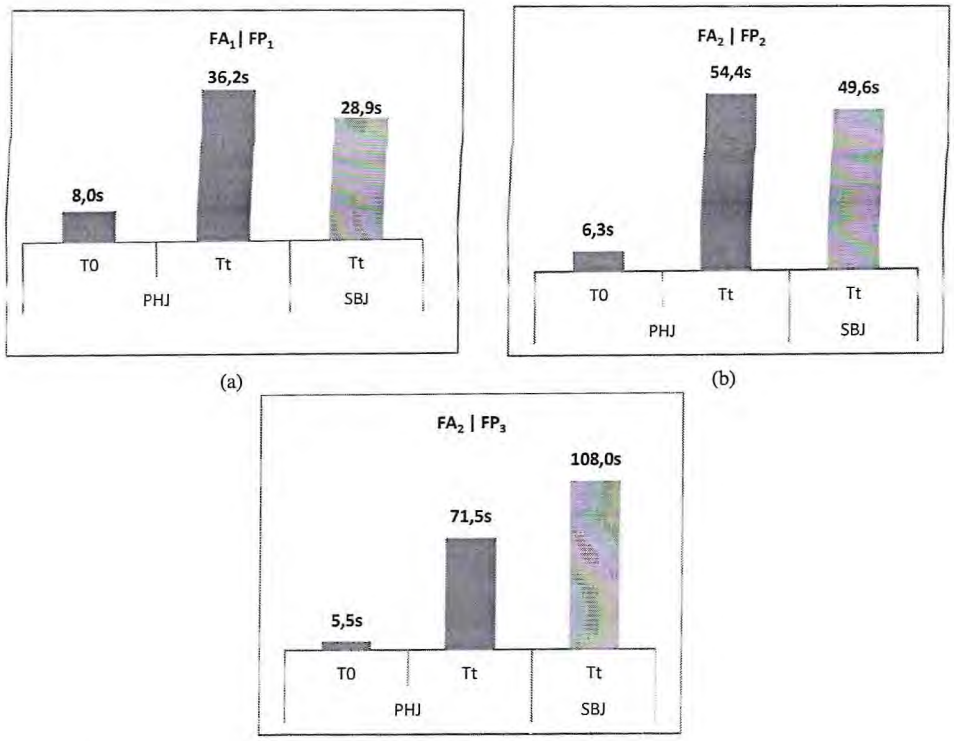
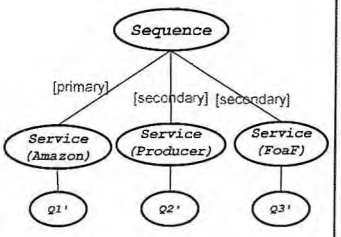


Figura 68 – Gráficos de comparação dos resultados das consultas $Q_{2.1}$

Consulta $Q_{2.2}$ – Análise da Estratégia Adaptativa

O principal objetivo deste conjunto de experimentos é testar a estratégia adaptativa comparando-a com diversas combinações de planos de execução previamente definidos. Assim, foi considerada parte do plano gerado pelo algoritmo de reformulação ilustrado na Figura 69(a); sendo as subconsultas Q_1', Q_2' e Q_3' apresentadas na Figura 69(b).



Q_1' (Amazon)

```
SELECT ?t ?cm ?rt ?rvr
?pdr
WHERE {
 ?r rdf:type a:Review.
 ?r a:title ?cm .
 ?r a:rating ?rt .
 FILTER (?rt > 8 )
 ?r a:reviewer ?rvr .
 ?r a:reviewFor ?p .
 ?p rdfs:label ?t .
 ?p a:producer ?pdr.
}
```

Q_2' (Producer)

```
SELECT ?pdr
WHERE {
 ?pdr p:pCountry ?cy .
 FILTER (?cy, 'USA')
}
```

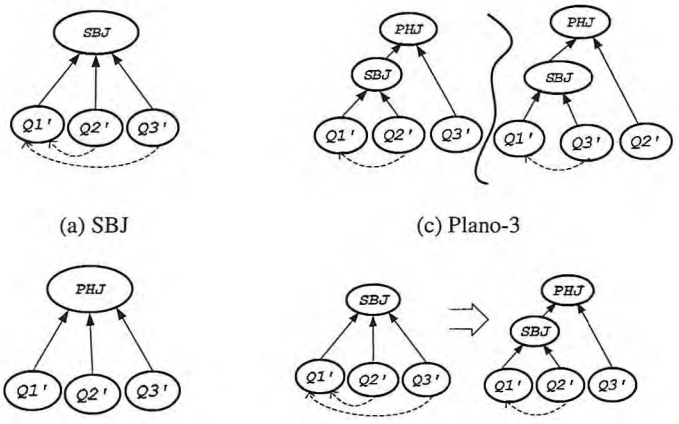
Q3' (Foaf)

```
SELECT ?rvr
WHERE {
 ?rvr f:country ?ct .
 FILTER (?ct, 'FR')
}
```

(b)

Figura 69 – Resultado do algoritmo de reformulação sobre a consulta $Q_{2.2}$

Na Figura 70 apresentam-se alguns possíveis planos de execução da consulta $Q_{2.2}$. Os planos da Figura 70(a) e 72(b) utilizam apenas os algoritmos SBJ e PHJ, respectivamente. O plano da Figura 70(c) executa em paralelo *Q2'* e *Q3'*, sendo que, para a consulta que terminar primeiro aplica-se o SBJ com *Q1'* e a outra consulta (*Q2'* ou *Q3'*) continua executando e as tuplas enviadas ao mediador serão executadas com algoritmo PHJ. Por fim, tem-se a estratégia adaptativa ilustrada no plano da Figura 70(d), que inicia a execução com algoritmo SBJ, observou-se que a quantidade de *binds* recuperados pela consulta *Q3'* ultrapassou o limite máximo (*TAM_MAXIMO*), que é condição suficiente para mudar para o algoritmo PHJ, conforme ilustrado no plano do lado direito da Figura 70(d).



(a) SBJ (c) Plano-3

(b) PHJ (d) Adaptativo

Figura 70 – Diferentes planos de execução para a consulta $Q_{2.2}$

Analisando-se os resultados sintetizados na Figura 71, ambos os algoritmos SBJ e PHJ apresentaram desempenho pior, enquanto que o plano-3 e o plano adaptativo, que combinam os benefícios dos dois algoritmos apresentaram um desempenho global melhor.

Observe que a estratégia adaptativa inicia com o plano da Figura 70(a) e durante a execução, a consulta Q_3' ultrapassa a quantidade máxima de *binds* e muda para o PHJ, essa mudança de plano é refletida na Figura 70(d).

O emprego da estratégia adaptativa se mostra adequada para flexibilizar o emprego dos algoritmos SBJ e PHJ, como demonstrado pelos experimentos referente à consulta $Q_{2.2}$. Os resultados obtidos motivam a investigação adicional a fim de ampliar a estratégia adaptativa para lidar, por exemplo, com problemas de tolerância a falhas.

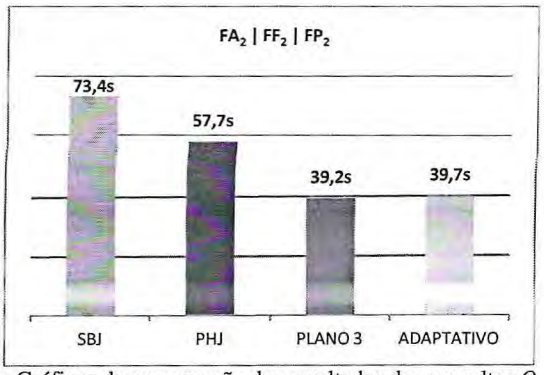


Figura 71 – Gráficos de comparação dos resultados das consultas $Q_{2.2}$

9.3 Considerações do capítulo

Neste capítulo foram descritos os estudos sobre o desempenho e o comportamento dos algoritmos de junção PHJ e SBJ e a estratégia adaptativa, sendo analisados os resultados desses estudos através de experimentos. Os resultados provêem respostas para algumas perguntas que esta tese procura equacionar, colocadas no Capítulo 1, fornecendo evidências empíricas da escalabilidade da abordagem de otimização e execução de consultas para integração de dados no padrão de *Linked Data*.

Inicialmente, observou-se que o tamanho dos *binds* (parâmetro *t*) afeta o tempo do processamento da operação de junção. Quando o algoritmo tem comportamento semelhante ao *bind-join* (*t=1*), constatou-se perda significativa de desempenho. Como observado nos experimentos, esse efeito é mais grave quando o total de *binds* recuperados for maior.

Ademais, evidenciou-se que os algoritmos SBJ e PHJ atendem a objetivos diferentes e podem ser combinados em uma estratégia adaptativa para explorar as melhores características de cada um. Como foi mostrado em comparação com diversos planos de execução, a estratégia adaptativa atingiu o objetivo de obter bom desempenho combinando simultaneamente os benefícios do SBJ e PHJ.

A desvantagem da solução proposta é a parametrização empírica definida no parâmetro *t* a partir de experimentos e das constantes utilizadas na estratégia adaptativa. A pergunta natural é "como definir parâmetros ótimos", isto é, parâmetros que produzam o melhor desempenho em todos os casos. Essa é uma questão para trabalhos futuros. No entanto, essa interrogação deixa espaço para pensar que, se os parâmetros definidos nos experimentos não são os parâmetros ótimos, então resultados ainda melhores do que os apresentados neste capítulo podem ser obtidos.

10 CONCLUSÕES E TRABALHOS FUTUROS

10.1 Considerações Finais

A área de pesquisa deste trabalho é a integração de dados no padrão de *Linked Data*. Apresentou-se, nessa perspectiva, um *framework* que propunha a revelar-se mais que um simples ambiente de mediação, para tanto, aproveitando-se da arquitetura de três níveis de esquema para simplificar e tornar mais eficiente a integração de dados no padrão de *Linked Data,* pois as ontologias de aplicação são um fragmento homogêneo da ontologia de domínio e nenhuma inferência é necessária.

As ontologias de aplicação ajudaram a dividir o problema de processamento de consultas em dois subproblemas (menos desafiadores): (a) responder a consultas no âmbito de fragmentos homogêneos, isto é, como reformular uma consulta sobre a ontologia de domínio em um ou mais subconsultas expressas em termos das ontologias de aplicação, (b) como responder a consultas no contexto de *data-exchange*, ou seja, como transformar subconsultas sobre uma ontologia de aplicação em uma consulta sobre as fontes de dados.

Apresentou-se, então, um método para avaliar eficientemente consultas SPARQL distribuídas que definisse as etapas de tradução; reformulação da consulta; e execução e composição do resultado final. Destacaram-se, nesse processo, as etapas de reformulação de consulta e execução, que constituem as principais contribuições deste trabalho.

O algoritmo de reformulação possui duas características importantes: consultar apenas fontes de dados que podem contribuir com qualquer resultado intermediário, sem precisar recorrer a mecanismos de inferência para fazer a expansão da consulta; utiliza-se ligações *same-as* e *URI-links* durante a reformulação da consulta, visando minimizar o problema de informação incompleta e, consequentemente, aumentar o valor da informação no momento em que dados de várias fontes são acessados, relacionados e combinados.

Na etapa de execução, foram propostos algoritmos para junção que exploram imprevisibilidade do tempo de resposta dos SPARQL *endpoints*. Definiu-se o algoritmo *set-bind-join*, que objetiva a redução na transmissão de dados intermediários para o mediador. Esse algoritmo explora o paralelismo intraoperador para melhorar a sua eficiência. Revisitou-se também uma estratégia de junção paralela baseada no algoritmo *pipelining hash-join* para

banco de dados paralelos, que permite produzir resultados de maneira incremental à medida que ficam disponíveis.

Por fim, foi definida uma estratégia adaptativa simples, mas efetiva, para o processamento de consulta, a qual viabiliza intercambiar os algoritmos de junção em situações de atraso na chegada dos dados provenientes dos SPARQL *endpoints* ou quando o tamanho do conjunto de *binds* seja tão grande que não vale a pena continuar executando o algoritmo *set-bind-join*.

Cabe ressaltar que, apresentaram-se regras de correção que indicam que o algoritmo proposto de reformulação de consulta reescreve corretamente as consultas sobre as fontes de dados. A etapa de execução e consolidação dos resultados foi validada através de experimentação e os resultados confirmam a intuição de que a combinação dos algoritmos de junção com uma estratégia adaptativa de execução de consultas distribuídas efetivamente capazes de fornecer resultados satisfatórios.

Assim, conclui-se que o presente trabalho conseguiu atingir seus objetivos, fornecendo uma solução não-intrusiva e de fácil utilização para processamento de consultas em um ambiente mediado no contexto de dados publicados no padrão de *Linked Data* integração de dados no padrão de *Linked Data* que visa a preencher uma importante lacuna em função da carência de trabalhos relacionados.

Até a presente data, foram geradas as seguintes publicações a partir dos resultados obtidos durante o desenvolvimento desta tese:

a) Pinheiro, J. C., Vidal, V. M. P. (2009) *Efficient Query Processing in an Ontology-Based Mediation System.* In Proceedings of WTDBD 2009, Fortaleza – CE -Brazil, 2009.

b) Pinheiro, J. C.,Vidal, V. M. P, Sacramento, E.R., Macedo, J.A.F. (2009). *An Ontology-Based Framework for Heterogeneous Data Integration.* Poster published in Proceedings of ER 2009, Gramado-RS-Brazil, 2009.

c) Pinheiro, J. C., Vidal, V. M., Macedo, J. A., Sacramento, E. R., Casanova, M. A., & Porto, F. M. (2010). *Query processing in a three-level ontology-based data integration system.* In Proceedings of the 12th International Conference on

Information Integration and Web-based Applications & Services (iiWAS '10). ACM, New York, NY, USA, 283-290. http://doi.acm.org/10.1145/1967486.1967532

d) Vidal, V. M., Macedo, J. A., Pinheiro, J. C., Casanova, M. A., Porto, F. M. (2011). *Query Processing in a Mediator Based Framework for Linked Data Integration.* International Journal of Business Data Communications and Networking (IJBDCN). IJBDCN @ IGI Global. Volume 7(2): 29-47.

Como trabalhos adicionais, durante o período do doutorado, também foram publicados:

a) Pinheiro, J. C.,Vidal, V. M. P, Sacramento, E. R., Lóscio, B. F. *Serviços Web Semântico.* ERCEMAPI, São Luís-MA-Brasil, 2008.

b) Teixeira, G. M., Vidal, V. M. P., Pinheiro, J. C. *Um Enfoque para Reformulação Eficiente de Consultas sobre Visões XML de Integração de Dados.* ERCEMAPI, São Luís-MA-Brasil, 2008.

c) Teixeira, G.M., Vidal, V. M. P., Pinheiro, J. C. *Processamento eficiente de consultas em um sistema de mediação baseado em XML.* InfoBrasil, Fortaleza-CE-Brasil, 2009.

d) Teixeira, G. M., Pinheiro, J. C., Lemos, F., Vidal, V. M. P. *Um Framework para Sistemas de Integração de Dados Baseados em XML.* Poster published in Proceedings of Simpósio Brasileiro de Bancos de Dados, Fortaleza-CE-Brazil, 2009.

10.2 Trabalhos futuros

A integração de dados e, em especial, o processamento distribuído de consultas é um assunto bastante complexo e as diferentes etapas do processamento de uma consulta em ambiente distribuído podem ser ampliadas de diversas formas. A seguir, são comentadas algumas sugestões de melhorias ou trabalhos futuros que podem ser desenvolvidos a partir

dos resultados obtidos. Estas sugestões servem também para demonstrar que o desenvolvimento desta tese abre uma linha de pesquisa ampla, permitindo seu prosseguimento através de trabalhos de iniciação científica, dissertações de mestrado e, possivelmente, outras teses de doutorado.

Como exemplo de melhorias ou trabalhos futuros que podem desenvolver a partir dos resultados obtidos, tem-se:

a) Ampliar o desenvolvimento do *framework* e do método de processamento distribuído de consultas. Neste sentido, uma continuação interessante deste trabalho seria a implementação de todas as etapas do método de processamento de consulta.
b) Para ajudar o usuário na tarefa de definir consulta SPARQL, uma interface gráfica pode ser desenvolvida permitindo ao usuário intuitivamente formular uma consulta usando um vocabulário de domínio bem conhecido. O usuário navega através de conceitos e propriedades da ontologia de domínio e escolhe o conceito primário a ser consultado, estabelece algumas restrições sobre suas propriedades (cláusula *where*) e, finalmente, seleciona as propriedades a serem recuperadas (cláusula *select*).
c) Um problema atual é que os SPARQL *endpoints* falham com certa frequência, principalmente devido a problemas de congestionamentos, então, é importante o fornecimento de mecanismos de tolerância a falhas. Para modelar este ambiente, falhas nas consultas podem ser incorporadas aos experimentos com o objetivo de simular situações reais, em que um SPARQL *endpoint* está ocupado e não pode responder (temporariamente indisponível) ou um serviço é interrompido e os dados devem ser recuperados de uma fonte de dados replicada.
d) Um teorema que estabeleça a relação matemática entre os objetivos de maximizar o processamento paralelo com o algoritmo *pipelining hash join*, com o objetivo de reduzir o envio de resultados intermediários através do algoritmo *set-bind-join*. A partir desta relação, possivelmente, poder-se-á construir uma estratégia mais precisa para otimização adaptativa que conduza a um melhor desempenho.
e) Correlacionar e testar, no algoritmo *set-bind-join*, a quantidade de consultas que incluam os *binds* em computadores com processadores de 2, 4 e 8 núcleos.
f) Utilizar o modelo de consulta do *eddy*, que é responsável por ler tuplas de uma

fonte de dados e distribuí-las para os demais operadores do plano, que as processarão e as devolverão para serem mapeados ao próximo operador. Para tanto, será preciso definir uma política de roteamento que determina para quais operadores uma tupla pode ser encaminhada e uma política de controle de fluxo determina para qual destes operadores a tupla deve seguir.

g) Elaboração de uma proposta para definição de uma função de custo para determinação do melhor plano de execução inicial, quando se tiver estatísticas de qualidade das fontes de dados. Para fontes de dados no padrão de *Linked Data*, pode-se utilizar o *RDF-stats* (LANGEGGER; WOSS, 2009), um subprojeto do SemWIQ (Langegger, 2010) desenvolvido para coletar automaticamente informações como, cadinalidade, histograma. É importante destacar que conforme dito em Langegger (2010, p196), o uso do RDF-Stats é viável se a fonte de dados for relativamente pequena ou se ele está situado na mesma rede local do mediador. Obermeier e Nixon (2008) descrevem um modelo de custo que se destina a atuar como um subcomponente de um otimizador de consulta para o processamento de consultas SPARQL distribuídas, servindo como indicador de custo para outros subcomponentes do otimizador para escolha de algoritmos de junção e sua ordem de execução. Os resultados desses dois trabalhos (LANGEGGER; WOSS, 2009; OBERMEIER; NIXON, 2008) podem ser futuramente aproveitados em uma estratégia de otimização proposta para auxiliar na geração mais precisa de um plano de execução inicial. Atualmente, considera-se somente a geração do plano inicial na ausência de estatísticas.

h) Aumento do escopo da gramática da linguagem SPARQL suportada pelo Mediador, como a inclusão de consultas do tipo *CONSTRUCT*, tratamento da cláusula *OPTIONAL* e funcionalidade da nova versão 1.1, tipo o uso de funções de agregação, utilização da cláusula *GROUP BY* e *HAVING*.

i) A fusão de dados pode ser implementada como uma etapa de pós-processamento da consulta, intencionando, dessa maneira, melhorar a qualidade dos dados recuperados.

Como é possível constatar, são várias as direções em que se pode expandir e melhorar o trabalho apresentado, cujos resultados têm possibilidade de aplicação imediata, o que, por sinal, era um objetivo que, também, tinha-se em mente ao iniciá-lo.

REFERÊNCIAS

AMANN, B.; BEERI, C.; FUNDULAKI, I.; SCHOLL, M. **Ontology-Based Integration of XML Web Resources**. Proceedings of the First International Semantic Web Conference on The Semantic Web, Springer-Verlag, 2002, 117-131.

ANGLES, R.; GUTIERREZ, C. **The Expressive Power of SPARQL**. Proceedings of the 7th International Conference on the Semantic Web, Springer-Verlag, 2008, 114-129.

ARENAS, M.; PEREZ, J. **Querying Semantic Web Data with SPARQL**. In Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems (PODS '11). ACM, New York, NY, USA, 305-316.

ARQ. (2008). **SPARQL query processing framework**. Hewlett-Packard Development Company. Disponível em < http://jena.sourceforge.net/ARQ/>. Acesso em: 10 mar. 2011.

ATRE, M.; CHAOJI, V.; ZAKI, M. J.; HENDLER, J.A. **Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data**. Proceedings of the 19th international conference on World wide web, ACM, 2010, 41-50.

AUER, S.; DIETZOLD, S.; LEHMANN, J.; HELLMANN, S.; AUMUELLER, D. **Triplify: light-weight Linked Data publication from relational databases**. Proceedings of the 18th international conference on World wide web, ACM, 2009, 621-630.

AVNUR, R.; HELLERSTEIN, J. M. **Eddies: continuously adaptive query processing**. SIGMOD Rec.; 29, 261-72, 2000.

AYRES, F. V. M; PORTO, FÁBIO; MELO, R. N. U**ma máquina extensível para suporte a novos modelos de execução de consultas**. In: XVIII Simposio Brasileiro de Banco de Dados, Anais do XVIII Simposio Brasileiro de Banco de Dados. Manaus : UFAM, 2003.

BARU, C.; GUPTA, A.; LUDASCHER, B.; MARCIANO, R.; PAPAKONSTANTINOU, Y.; VELIKHOV, P. **XML-based information mediation with MIX**. SIGMOD Rec.; ACM, 28, 597-9, 1999.

BECHHOFER, S.; HARMELEN, F.V.; HENDLER, J.; HORROCKS, I.; McGUINNESS, D.; PATEL-SCHNEIJDER, P. **OWL Web Ontology Language Reference**. World Wide Web

Consortium (W3C). W3C. 2004. Disponível em < http://www.w3.org/TR/owl-ref/>. Acesso em: 8 set. 2010.

BECKETT, D.; BROEKSTRA, J. **SPARQL Query Results XML Format**. W3C. 2006. *Disponível em < http://www.w3.org/TR/2006/CR-rdf-sparql-XMLres-20060406/>*. Acesso em: 19 maio. 2011.

BERNERS-LEE, T.; FIELDING, R.; MASINTER, L. **Uniform resource identifier (URI): Generic syntax**. Internet Engineering Task Force RFC 3986, Internet Society (ISOC). 2005 Disponível em <http http://tools.ietf.org/html/rfc3986>. Acesso em 17 mar. 2010.

BERNERS-LEE, T.; CHEN, Y.; CHILTON, L.; CONNOLLY, D.; DHANARAJ, R.; HOLLENBACH, J. et al. **Tabulator: exploring and analyzing linked data on the semantic web**. Proceedings of the 3rd International Semantic Web User Interaction, 2006.

BIZER, C.; CYGANIAK, R.. **D2R Server–publishing relational databases on the semantic web**. Poster at the 5th International Semantic Web Conference, 2006.

BIZER, C. & CYGANIAK, R. **Publishing Relational Databases on the Web as SPARQL-Endpoints**. 2006. 15th International World Wide Web Conference (WWW2006), 2006

BIZER, C.; SCHULTZ, A. **Benchmarking the performance of storage systems that expose SPARQL endpoints**. In Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledgebase, 2008.

BIZER, C.; HEATH, T.; BERNERS-LEE, T. **Linked Data – The Story So Far**. International Journal on Semantic Web and Information Systems, 5(3), 1-22, 2009.

BIZER, C., JENTZSCH, A., and CYGANIAK, R. **State of the LOD Cloud.** 2011. Disponível em < http://www4.wiwiss.fu-berlin.de/lodcloud/state/ >.

BRICKLEY, D.; GUHA, R.V. **RDF Vocabulary Description Language 1.0: RDF Schema**. W3C Recommendation. 2004. Disponível em < http://www.w3.org/TR/2004/REC-rdf-schema-20040210/>. Acesso em: 8 set. 2010.

BUIL-ARANDA, C.; ARENAS, M.; CORCHO, O. **Semantics and Optimization of the SPARQL 1.1 Federation Extension**. Heraklion: ESWC'11, 2011.

CALVANESE, D.; GIACOMO, G. D.; LEMBO, D.; LENZERINI, M.; POGGI, A.; ROSATI, R. et al. **Data Integration through DL-LiteA Ontologies**. In 3rd Int. Workshop on Semantics in Data and Knowledge Bases (SDKB 2008), Springer, 2008, 4925, 26-47.

CALVANESE, D.; GIACOMO, G. D.; LENZERINI, M.; LEMBO, D.; POGGI, A.; ROSATI, R. **MASTRO-I: Efficient Integration of Relational Data through DL Ontologies**. In Description Logics. pp. 227-34, 2007.

CARROLL, J.J.; BIZER, C.; HAYES, P.; STICKLER, P. **Named graphs, provenance and trust**. WWW '05: Proceedings of the 14th international conference on World Wide Web, ACM, 2005, 613-622.

CARROLL, J.J.; DICKINSON, I.; DOLLIN, C.; REYNOLDS, D.; SEABORNE, A.; WILKINSON, K. **Jena: implanting the semantic web recommendations**. Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters, ACM, 2004, 74-83.

CLARK, K.G.; FEIGENBAUM, L.; TORRES, E. **SPARQL Protocol for RDF**. W3C. 2008. Disponível em < http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115>. Acesso em: 20 maio. 2011. 2008.

CRUZ, I. F.; XIAO, H.; HSU, F. **An Ontology-Based framework for XML semantic integration**. In proceedings of the International Database Engineering and Applications Symposium, IEEE Computer Society, 2004, 217-226.

CRUZ I., XIAO H. **The role of ontologies in data integration**. Journal of Engineering Intelligent Systems", vol. 13, num. 4, 2005.

ESSID, M.; BOUCELMA, O.; COLONNA, F.-M.; LASSOUED, Y. **Query processing in a geographic mediation system**. Proceedings of the 12th annual ACM international workshop on Geographic information systems, ACM, 2004, 101-108.

FENSEL, D. **Ontologies: a silver bullet for knowledge management and electronic commerce**. Springer-Verlag New York, Inc. 2001.

FERNANDEZ, M. F.; MORISHIMA, A.; SUCIU, D.; TAN, W. C. **Publishing Relational Data in XML: the SilkRoute Approach**. IEEE Data Eng. Bull. 24(2), 12-19, 2001.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. **Design patterns: elements of reusable object-oriented software**. Addison-Wesley Longman Publishing Co.; Inc., 1995.

GHIDINI, C.; SERAFINI, L. **Reconciling concepts and relations in heterogeneous ontologies**. Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), Springer, 2006, Volume 4011/2006, 50-64.

GRAEFE, G. **Query evaluation techniques for large databases**. ACM Comput. Surv.; 25, 73-169, 1993.

GROPPE, J.; GROPPE, S.; Bers, S.; LINNEMANN, V. **Efficient processing of SPARQL joins in memory by dynamically restricting triple patterns**. Proceedings of the 2009 ACM symposium on Applied Computing, ACM, 2009, 1231-1238.

GRUBER, T.R. **A translation approach to portable ontologies**. Knowledge Acquisition, Acquis., Academic Press Ltd., 1993, 5, 199-220.

HAAS, L.M.; KOSSMANN, D.; WIMMERS, E.L.; YANG, J. **Optimizing queries across diverse data sources**. Proceedings of the 23rd International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 1997, 276-285.

HALEVY, A. Y. **Theory of answering queries using views**. SIGMOD Rec. 29, 4, Dec 2000, 40-47. http://doi.acm.org/10.1145/369275.369284

HALEVY, A. Y; RAJARAMAN, A.; ORDILLE, J. **Data integration: the teenage years**. Proceedings of the 32nd international conference on Very large data bases, VLDB Endowment, 2006, 9-16.

HARTIG, O.; BIZER, C.; FREYTAG, J.-C. **Executing SPARQL queries over the Web of Linked Data**. Proceedings of the 8th International Semantic Web Conference, Springer-Verlag, 2009, 293-309.

HARTIG; LANGEGGER. **A Database Perspective on Consuming Linked Data on the Web**. Datenbank-Spektrum, 2010, 14(2):1–10.

HELLERSTEIN, J. M.; FRANKLIN, M. J.; CHANDRASEKARAN, S.; DESHPANDE, A.; HILDRUM, K.; MADDEN, S.; RAMAN, V. & SHAH, M. A. **Adaptive query processing: technology in evolution**. IEEE Data Eng. Bull., 2000, 23, 7-18.

HULL, R.; YOSHIKAWA, M. **ILOG: declarative creation and manipulation of object identifiers**. Proceedings of the sixteenth international conference on Very large databases, Morgan Kaufmann Publishers Inc., 1990, 455-468.

HUNT, C. **TCP/IP network administration**. 3ª ed. O'Reilly Networking. O'Reilly Media, Inc., 2002.

IVES, Z.G.; HALEVY, A.Y.; WELD, D.S. **Adapting to source properties in processing data integration queries**. Proceedings of the 2004 ACM SIGMOD international conference on *Management of data, ACM, 2004, 395-406.*

JENA/TDB. (2010). **Jena TDB**. Disponível em <http://jenawiki.hpl.hp.com/wiki/TDB >. Acesso em 29 mai. 2011.

KLYNE, G.; CARROLL, J. J. **Resource Description Framework (RDF): Concepts and Abstract Syntax**. W3C Recommendation, World Wide Web Consortium. 2004. Disponível em < http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/>. Acesso em 07 mar. 2010.

KOSSMANN, D. **The state of the art in distributed query processing**. ACM Comput. Surv., ACM, 2000, 32, 422-469.

KRUK, S. R. **FOAF-Realm: control your friends access to the resource**. 2004. FOAF Workshop, 2004.

LANGEGGER, A. **A flexible architecture for virtual information integration based on Semantic Web Concepts**. Phd Thesis. University Linz: University Linz, 2010. Disponível em < http://www.langegger.at/papers/Langegger-phd-thesis.pdf/>. Acesso em 07 mar. 2011.

LANGEGGER, A.; WOSS, W. **RDFStats – an extensible RDF statistics generator and library**. Database and Expert Systems Applications, International Workshop on, IEEE Computer Society, 2009, 0, 79-83.

LANGEGGER, A.; WOSS, W.; BLOCHL, M. **A semantic web middleware for virtual data integration on the web**. Proceedings of the 5th European semantic web conference on The semantic web: research and applications, Springer-Verlag, 2008, 493-507.

LAWRENCE, R. **Early hash join: a configurable algorithm for the efficient and early production of join results**. Proceedings of the 31st international conference on Very large data bases, VLDB Endowment, 2005, 841-852.

LEHTI, P.; FANKHAUSER, P. **XML data integration with OWL: experiences and challenges**. Applications and the Internet, IEEE/IPSJ International Symposium on, IEEE Computer Society, 2004, 0, 160.

LENZERINI, M. **Data integration: a theoretical perspective**. Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, ACM, 2002, 233-246.

LIBKIN, L.; SIRANGELO, C. **Data exchange and schema mappings in open and closed worlds**. J. Comput. Syst. Sci., Academic Press, Inc., 2011, 77, 542-571.

LUTZ, M. **Ontology-based discovery and composition of geographic information services**. Institute for Geoinformatics, University of Münster, Germany, 2005.

MANOLESCU, I.; FLORESCU, D.; KOSSMANN, D. **Answering XML queries on heterogeneous data sources.** Proceedings of the 27th International Conference on Very Large Data Bases, Morgan Kaufmann Publishers Inc., 2001, 241-250.

MAGALHÃES, R. P.; SALAS, P. R.; VIDAL, V. M. P.; MACEDO, J. A. F; BREITMAN, K.; CASANOVA, M. A. **Linked Data: construindo um espaço de dados global na Web**. Minicurso a ser apresentado noXXVI SBBD, Florianopolis, Santa Catarina – BR, 2011.

MENA, E.; KASHYAP, V.; SHETH, A.; ILLARRAMENDI, A. **OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies**. IEEE Computer Society Press, 1996, 14-25.

NECIB, B. **Ontology-based semantic query processing in database systems**. Phd Thesis. Humboldt Universität. 2007.

NEUMANN, T.; WEIKUM, G. The **RDF-3X engine for scalable management of RDF data**. The VLDB Journal, Springer-Verlag New York, Inc., 2010, 19, 91-113.

NOY, N.F. **Semantic integration: a survey of ontology-based approaches**. SIGMOD Rec., ACM, 2004, 33, 65-70.

NOY, N.F.; McGuinness, D.L. **Ontology development 101: a guide to creating your first ontology**. Stanford Knowledge Systems Laboratory, Stanford. 2001.

OBERMEIER, Philipp; NIXON, Lyndon. **A cost model for querying distributed rdf-repositories with sparql**. In Proceedings of the Workshop on Advancing Reasoning on the Web: Scalability and Commonsense Tenerife, Spain, 2008.

OLAF, Gorlitz; STAAB, Steffen. **Federated Data Management and Query Optimization for Linked Open Data**. In New Directions in Web Data Management. Springer, 2011.

OZSU, M.T.; VALDURIEZ, P. ***Principles of distributed database systems.*** Prentice-Hall, Inc., 1999

PÉREZ, J.; ARENAS, M.; GUTIERREZ, C. **Semantics and complexity of SPARQL**. ACM Transaction. Database System, ACM, 2009, 34, 16:1-16:45.

PINHEIRO, J. C., VIDAL, V. M. P. **Efficient Query Processing in an Ontology-Based Mediation System**. In Proceedings of WTDBD 2009, Fortaleza – CE – Brazil, 2009.

PINHEIRO, J. C.,VIDAL, V. M. P, SACRAMENTO, E.R., MACEDO, J. A. F. (2009). **An Ontology-based framework for heterogeneous data integration**. Poster published in Proceedings of ER 2009, Gramado - RS - Brazil, 2009.

PINHEIRO, J.C.; VIDAL, V. M.. P; MACEDO, J.A.; SACRAMENTO, E.R.; CASANOVA, M. A., PORTO, F. M. **Query processing in a three-level ontology-based data integration system**. In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS '10). ACM, New York, NY, USA, 283-290.

POGGI, A.; LEMBO, D.; CALVANESE, D.; GIACOMO, G. D.; LENZERINI, M.; ROSATI, R. **Linking data to ontologies**. In Spaccapietra, S. (ed.). Journal on data semantics X. pp. 133-73, 2008. Springer-Verlag.

PORTO, FÁBIO; AYRES, F. V. M; MELO, R. N. **QEEF: a framework for query execution engine**. In: Congresso Argentino de Ciências de La Comnputacion, 2002, Buenos Aires. Anais do CACIC2002, 2002. p. 75-86.

PRUD'HOMMEAUX, E. **Case study: FeDeRate for drug research**. In W3C (Ed.). W3C. 2004

PRUD'HOMMEAUX, E.; BUIL-ARANDA, C. **SPARQL 1.1 Federated Query**. W3C. 2011. Disponível em < SPARQL 1.1 Federated Query />. Acesso em 15 jun. 2011.

PRUD'HOMMEAUX, E.; SEABORNE, A. **SPARQL query language for RDF** (Working Draft). W3C. 2008. Disponível em <www.w3.org/TR/rdf-sparql-query/>. Acesso em 25 out. 2010.

QUILITZ, B.; LESER, U. **Querying distributed RDF data sources with SPARQL.** Proceedings of the 5th European semantic web conference on The semantic web: research and applications, Springer-Verlag, 2008, 524-538.

RAJARAMAN, A.; SAGIV, Y.; ULLMAN, J. D. **Answering queries using templates with binding patterns.** Proceedings of the fourteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, ACM, 1995, 105-112.

SACRAMENTO, E. R.; VIDAL, V. M. P.; MACEDO, J. A. F; LÓSCIO, B.F.; LIGIA, F.; CASANOVA, M. A. **Towards automatic generation of application ontologies.** Journal of Information and Data Management (JIDM), pp. 535-50, 2010.

SCHENK, S.; STAAB, S. **Networked graphs: a declarative mechanism for SPARQL rules, SPARQL views and RDF data integration on the web.** Proceeding of the 17th international conference on World Wide Web, ACM, 2008, 585-594.

SHETH, A.P.; LARSON, J.A. **Federated database systems for managing distributed, heterogeneous, and autonomous databases.** ACM Comput. Surv.; 22, 183-236, 1990.

STOCKER, M.; SEABORNE, A.; BERNSTEIN, A.; KIEFER, C.; REYNOLDS, D. **SPARQL basic graph pattern optimization using selectivity estimation.** Proceeding of the 17th international conference on World Wide Web, ACM, 2008, 595-604.

TEIXEIRA, G.M. **Um enfoque baseado em assertivas para reescrita de consultas sobre visões de integração de dados XML.** Dissertação de Mestrado. Fortaleza: UFC, 2009.

VIDAL, V.M.; SACRAMENTO, E.R.; MACÊDO, J.A.; CASANOVA, M.A. **An ontology-based framework for geographic data integration.** Proceedings of the ER 2009 Workshops (CoMoL, ETheCoM, FP-UML, MOST-ONISW, QoIS, RIGiM, SeCoGIS) on Advances in Conceptual Modeling - Challenging Perspectives, Springer-Verlag, 2009, 337-346.

VIDAL, V.M.; SANTOS, L.A.; OLIVEIRA, W.G.; LEMOS, F.C. **Um framework para publicação de dados armazenados em banco de dadosrelacional ou objeto relacional como XML.** In SBBD (ed.), Demo Session of 19th Brazilian Symposium on Databases. pp. 07-12. Brasília: SBBD, 2004.

VIDAL, V. M. P., SACRAMENTO E. R., JOSÉ A. F., CASANOVA M. A. **An Ontology-based framework for geographic data integration.** In: Proc. 3rd International Workshop on Semantic and Conceptual Issues in GIS (SeCoGIS 2009), in conjunction with the 28TH International Conference on Conceptual Modeling (ER 2009). Berlin / Heidelberg: Springer, 2009.

VIDAL, V. M., MACEDO, J. A., PINHEIRO, J. C., CASANOVA, M. A., PORTO, F. M. **Query processing in a mediator based framework for Linked Data integration**. International Journal of Business Data Communications and Networking (IJBDCN). IJBDCN @ IGI Global. Volume 7(2): 29-47, 2011.

WACHE, H.; VÖGELE, T.; VISSER, U.; STUCKENSCHMIDT, H.; SCHUSTER, G.; NEUMANN, H. et al. **Ontology-based integration of information – a survey of existing approaches**. pp. 108-17, 2001.

WIEDERHOLD, G. **Mediators in the architecture of future information systems**. Computer, IEEE Computer Society Press, 1992, 25, 38-49.

WILSCHUT, A.N.; APERS, P. **Dataflow query execution in a parallel main-memory environment**. Distrib. Parallel Databases, Kluwer Academic Publishers, 1993, 1, 103-128.

YU CONG; POPA LUCIAN. **Constraint-based XML query rewriting for data integration**. In Proceedings of the 2004 ACM SIGMOD international conference on Management of data. SIGMOD 2004. ACM, New York, NY, USA, 371-382. http://doi.acm.org/10.1145/1007568.1007611

ZEMÁNEK, J.; SCHENK, S. **Optimizing SPARQL queries over disparate RDF data sources through distributed semi-joins**. International Semantic Web Conference (Posters & Demos), CEUR-WS.org, 2008, 401.

ZIEGLER, P.; DITTRICH, K.R. **Three decades of data integration – all problems solved?** In 18th IFIP World Computer Congress (WCC 2004), Volume 12, Building the Information Society, Kluwer, 2004, 156, 3-12.